MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 139

MAC-M-357.
September 1967.

DECOMPOSITION OF A VISUAL SCENE INTO BODIES

Adolfo Guzmán

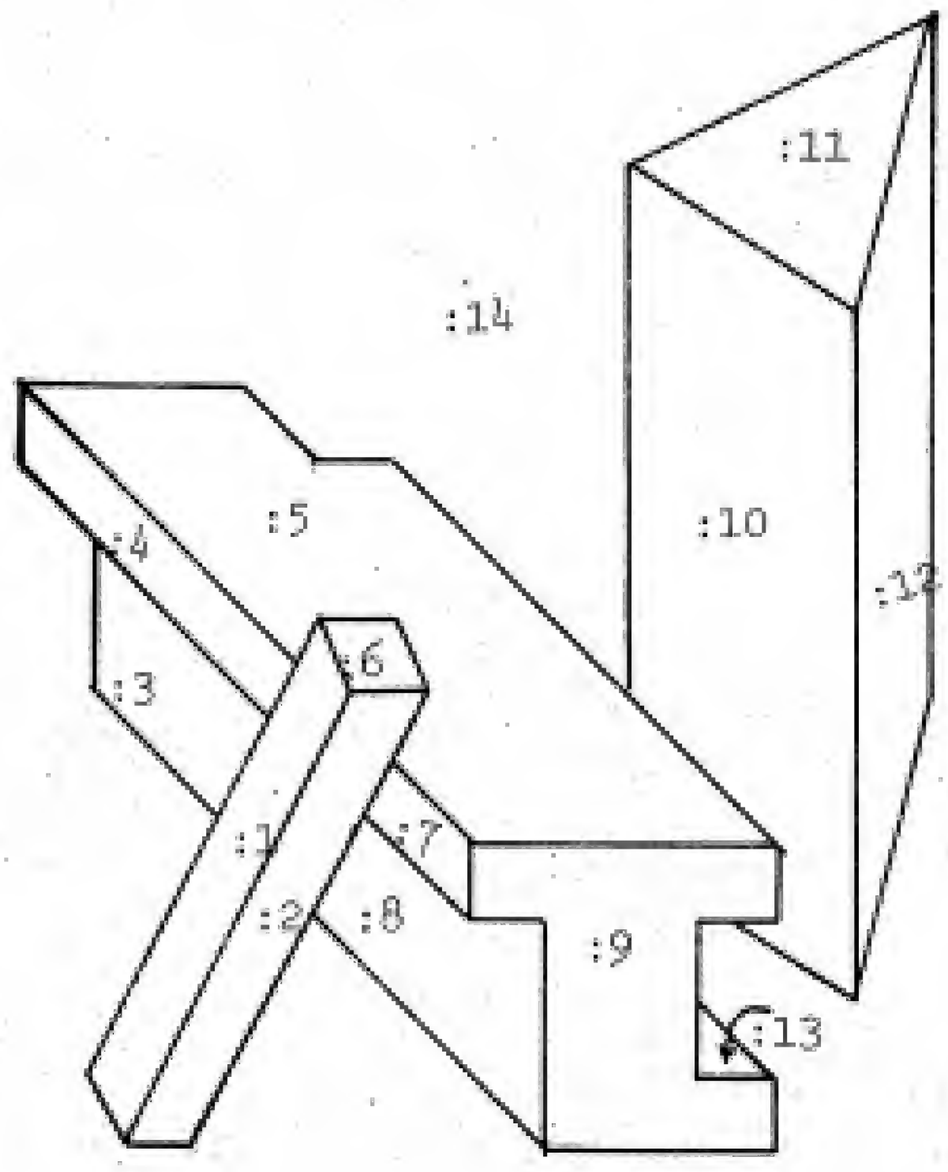


Figure 'T R I A L'.

The program analyzes this scene and finds 3 bodies:
(BODY 1 IS :1 :6 :2)
(BODY 2 IS :11 :12 :10)
(BODY 3 IS :4 :9 :5 :7 :3 :8 :13)

SECTION ONE: INTRODUCTION

This memorandum describes a program which finds bodies in a scene, presumably formed by 3-dimensional objects, with some of them perhaps not completely visible.

When SEE--the pretentious name of the program--analyzes the scene TRIAL (see figure 'TRIAL'), the results are

```
(BODY 1. IS :6 :2 :1)
(BODY 2. IS :11 :12 :10)
(BODY 3.  IS :4 :9 :5 :7 :3 :8 :13)
NIL
```

SEE looks for 3-dimensional objects in a 2-dimensional scene; the scene itself is not obtained from a visual input device, or from an array of intensities or brightness; rather, supposition is made that a pre-processing of some sort has taken place, and the scene to be analyzed is available in a symbolic format (to be described soon), in terms of points (vertices), lines (edges), and surfaces (regions).

SEE does not have a pre-conceived idea of the form or model of the objects which could appear in a given scene, the only supposition being that they are solid objects formed by plane surfaces; in this way, it can not find "cubes" or "houses" in a scene, since it does not know what a "house" is. Once SEE has partitioned a scene into bodies, some other program, probably a matcher, will work on them and decide which of those bodies are "houses."

Therefore, SEE forms a welcomed link between a pre-processor [1, 2] which transforms intensity pictures into point or line pictures (*), and a recognizer (such as TD [3] or DT [4]), which handles this line picture and finds bodies, objects or zones matching with certain patterns or models. Instead of searching through the whole scene looking for parts to match its models, the work of the recognizer (**) becomes simpler after SEE has partitioned the scene into bodies, because the data to be searched (matched) is smaller and better organized.

(*) A description of these processes in analysis of scenes is given in [5].

(**) Assuming the recognizer is interested in bodies which SEE can see (see [5]).

The analysis which SEE makes of the different scenes is generally acceptable, although in some ambiguous cases it behaves rather conservatively. Distributed over these pages, the reader will find examples of scenes analyzed by SEE, and the peculiarities and behavior of the program will become clear.

The program SEE is written in LISP, and has been tested in the PDP-6 machine of the Artificial Intelligence Group, Project MAC. A preliminary version was written in CONVERT, and was extensively used for quick test of ideas which shaped the program to its actual form.

SECTION TWO: INPUT FORMAT

The next section 'Format of a Scene', describes the form in which SEE expects the scene to be; eventually, this data will come from a visual input device (vidissector), through a pre-processor; nevertheless, for testing purposes, the scenes are entered by hand in a simplified format (called input format), and then some routines produce from it the necessary data for utilization by SEE. We will describe both in this section, which may be -- together with the next one -- skipped by the reader without much loss of continuity.

Example of a Scene: Suppose we want to describe the scene 'CUBE'. We begin by giving (in LISP) a value to 'CUBE'

```
(SETQ CUBE (QUOTE (A  1.0 1.0 (:1 B :4 G)
                   B  1.0 5.0 (:1 E :2 C :4 A)
                   C  3.0 7.0 (:2 D :4 B)
                   D  8.0 7.0 (:2 E :3 F :4 C)
                   E  6.0 5.0 (:2 B :1 G :3 D)
                   F  8.0 3.0 (:3 G :4 D)
                   G  6.0 1.0 (:1 A :4 F :3 E)
                   )))
```

Following each vertex, we give its coordinates and a list, counterclockwise ordered, of regions to which that vertex belongs and vertices to which that vertex is connected; in the next section this list receives the name 'KIND', and is more fully described.

The conversion of the scene, as just given, into the form which SEE expects, is made by the function LLENA; thus, (LLENA CUBE) will put in the property list of CUBE the properties REGIONS and VERTICES; in the property list of each vertex, the properties XCOR, YCOR, NREGIONS, NVERTICES and KIND; in the property list of each region, the properties NEIGHBORS and KVERTICES.

Then, we evaluate (FOOP (GET (QUOTE CUBE) (QUOTE REGIONS))), which puts the property FOOP to each region. All these properties are described in detail in the next section.

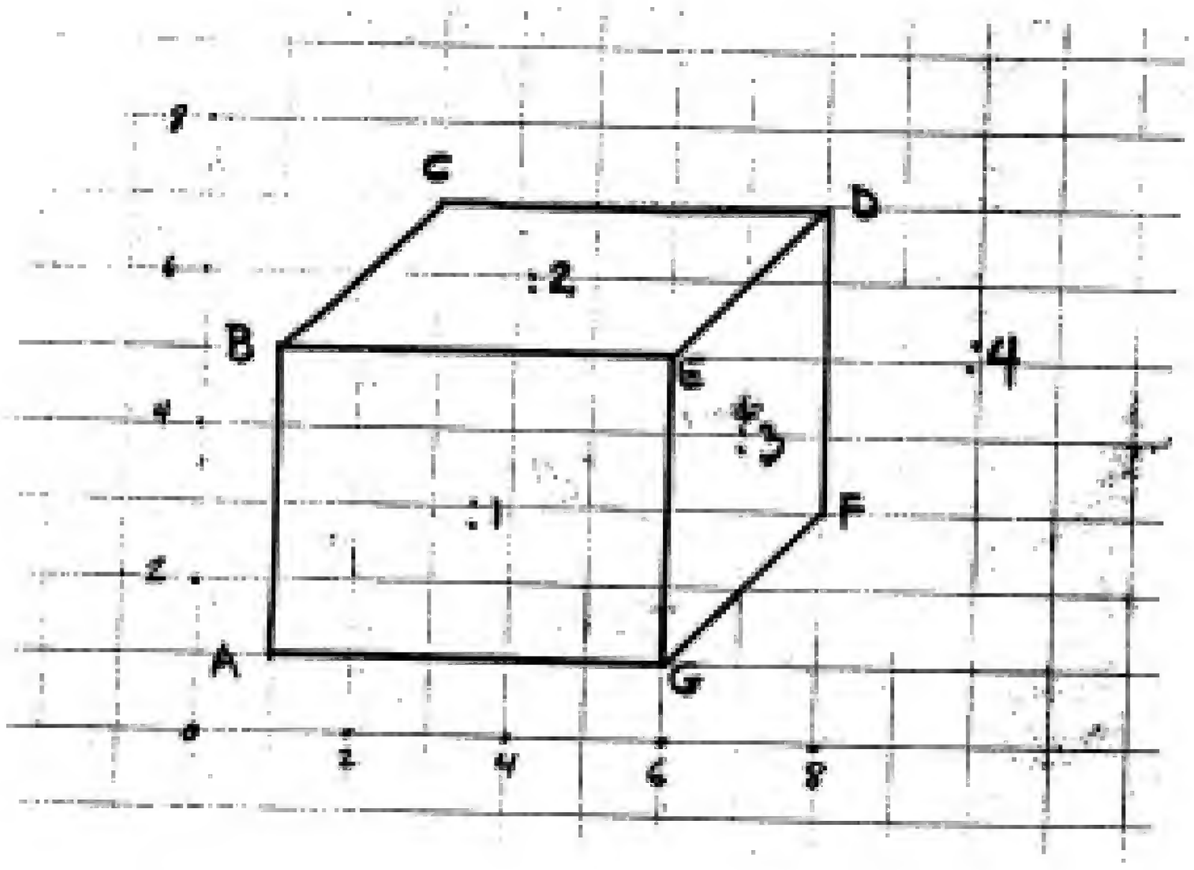


Figure 'C U B E'. A scene.

Finally, we perform

```
(MAPC (FUNCTION (LAMBDA (X)
                        (PUTPROP X (QUOTE BACKGROUND) (QUOTE BACKGROUND))))
      (PUTPROP (QUOTE CUBE) (QUOTE (:4)) (QUOTE BACKGROUND)))
```

We now give a more complete example. The following is the file TRIAL LISP; when loaded, it puts the scene TRIAL (see fig. 'TRIAL') in the form which SEE expects.

```
                                        (NOTISETQ TRIAL (QUOTE
(A 2.0 2.0 (#1 H #14 B) B 2.5 1.0 (#2 J #1 A #14 C)
C 3.5 1.0 (#2 B #14 D) D 5.0 4.0 (#2 C #14 CC #8 E)
E 5.5666666 5.3333333 (#8 EE #7 F #2 D)
F 6.0 5.0 (#7 FF #5 G #2 E) G 6.5 7.0 (#2 F #5 H #6 J)
H 5.0 8.0 (#5 I #6 G) I  5.0 8.0 (#1 J #6 H #5 K)
J 5.5 7.0 (#6 I #1 B #2 G) K 4.6666666 7.3333333 (#1 I #5 Q #4 L)
L 4.3333333 6.5666666 (#4 O #3 M #1 K)
M 3.56666666 5.33333333 (#3 N #14 A #1 L)
N 2.0 7.0 (#3 O #14 M) O 2.0 9.0 (#3 L #4 P  #14 N)
P 1.0 10.0 (#4 Q #14 O) Q 1.0 11.0 (#14 P #4 K #5 R)
R 4.0 11.0 (#5 S #14 Q) S 5.0 10.0 (#5 T #14 R)
T 6.0 10.0 (#5 J #14 S) U 9.0 7.0 (#10 LL #14 T #5 V)
V 11.0 5.0 (#10 U #5 FF #9 W) W 11.0 4.0 (#10 V #9 X)
X 10.5 4.0 (#10 W #9 Y #14 GG) Y 10.0 4.0 (#9 Z #14 X)
Z 10.0 3.0 (#9 KK #13 AA #14 Y)  AA 11.0 2.0 (#13 KK #9 BB #14 Z)
BB 11.0 1.0 (#9 CC #14 AA) CC 8.0 1.0 (#8 D #14 BB #9 DD)
DD 9.0 4.0 (#9 EE #8 CC) EE 7.0 4.0 (#7 E #8 DD #9 FF)
FF 7.0 5.0 (#7 EE #9 V #5 F) GG 12.0 3.0 (#12 JJ #10 X #14 HH)
HH 13.0 7.0 (#12 GG #14 II)
 II 13.0 16.0 (#11 JJ #12 HH #14 LL)
 JJ 12.0 12.0 (#11 LL #10 GG #12 II)
KK 10.0 2.0 (#13 Z #9 AA)  LL 9.0 14.0 (#10 JJ #11 II #14 U))
  )))

(DEFPROP INTER (LAMBDA (AX AY BX BY CX CY DX DY)
  (PROG (P Q X Y)
        (SETQ P (QUOTIENT (DIFFERENCE DY CY) (DIFFERENCE DX CX)))
        (SETQ Q (QUOTIENT (DIFFERENCE BY AY) (DIFFERENCE BX AX)))
        (SETQ X (PLUS (TIMES P CX) (MINUS CY) (MINUS (TIMES Q AX)) AY))
        (SETQ X (QUOTIENT X (DIFFERENCE P Q)))
        (SETQ Y (PLUS CY (TIMES P (DIFFERENCE X CX))))
        (RETURN (LIST X Y))))
 EXPR)

(DEFPROP LLENA (LAMBDA (A)
  (PROG (S R V)  M
        (COND ((NULL A) (PUTPROP SCENE R (QUOTE REGIONS))
                        (PUTPROP SCENE V (QUOTE VERTICES))  (RETURN 0)))
        (PUTPROP (CAR A) (CADR A)(QUOTE XCOR))
        (PUTPROP (CAR A)(CADDR A)(QUOTE YCOR))
        (PUTPROP (CAR A)(CADDDR A)(QUOTE KIND))
        (SETQ S
              (CONS (LIST (CAR A)
                          (PROG2 (PUTPROP (CAR A)  (CULL (CADDDR A))
                                          (QUOTE NREGIONS))
                                 (PUTPROP (CAR A)
                                          (CULL (CDR (CADDDR A)))
                                          (QUOTE NVERTICES))))
```

```
                    B))
          (CLNK (CADDDR A))  (SETQ V (CONS (CAR A) V))
          (MAPC (FUNCTION
                  (LAMBDA (A) (OR (MEMBER A R) (SETQ R (CONS A R)))))
                (CULL (CADDDR A)))
          (SETQ A (CDDDDR A))  (GO M)))
 EXPR)

(DEFPROP CULL (LAMBDA (A)
  (COND ((NULL A) A)  ((NULL (CDR A)) A)
        (T (CONS (CAR A)(CULL (CDDR A))))))
 EXPR)

(DEFPROP LNK (LAMBDA (A B N)
  (AND (CPROP B A N) (CPROP A B N)) )
 EXPR)

(DEFPROP CLNK (LAMBDA (B)
  (COND ((OR (NULL B)(NULL (CDR B))) B)
        ((NULL (CDDR B)) (CPROP (CAR B) (CADR B) (QUOTE KVERTICES))
                         (CPROP (CAR (CADDDR A))  (CADR B)
                                (QUOTE KVERTICES))
                         (LNK (CAR (CADDDR A))  (CAR B)
                              (QUOTE NEIGHBORS)))
        ((LNK (CAR B) (CADDR B) (QUOTE NEIGHBORS))
         (CPROP (CAR B) (CADR B) (QUOTE KVERTICES))
         (CPROP (CADDR B) (CADR B) (QUOTE KVERTICES))  (CLNK (CDDR B)))))
)
 EXPR)

  (DEFPROP IF (LAMBDA (L)
    (LIST (QUOTE COND)  (LIST (CADR L) (CADDR L))
          (CONS (QUOTE T) (CDDDR L))))
   MACRO)

  (DEFPROP CPROP (LAMBDA (A B C)
    (OR (MEMBER B (SETQ #: (GET A C))) (PUTPROP A (CONS B #:)C)))
   EXPR)

(SETQ SCENE (QUOTE TRIAL))

(SETQ TRI (LLENA TRIAL))

(DEFPROP *SAME* RONGA RONGA)

(MAPC (FUNCTION (LAMBDA
  (X) (PUTPROP X (QUOTE BACKGROUND)(QUOTE BACKGROUND))))
(PUTPROP (QUOTE TRIAL) (QUOTE (#1*)) (QUOTE BACKGROUND)))
```

```
(DEFPROP FOOP (LAMBDA (I R)
  (PROG2 (RPLACD (LAST (SETQ ** (REVERSE (FOOP* I)))) (LIST (CAR **)))
         (CDR **)))
 EXPR)

(DEFPROP FOOP* (LAMBDA (V)
  (COND ((EQ )
              (SETQ U
                    (CADR (SETQ **
                                (NEXT2OF (SETQ U (GET V (QUOTE <IND)))))))
))
         **)
        (T (NCONC ** (FOOP* U)))))
 EXPR)

(DEFPROP NEXT2OF (LAMBDA (M)
  (COND ((OR (NULL M) (NULL (CDR M))) (BREAK 1 NEXT2OF))
        ((NULL (CDDR M)) (COND ((EQ (CAR M) R) (LIST (CAR U) (CADR M)))
                               (T (BREAK 2 NEXT2OF))))
        ((EQ (CAR M) R) (LIST (CADDR M) (CADR M)))
        (T (NEXT2OF (CDDR M)))))
 EXPR)


(MAPC (FUNCTION (LAMBDA
  ((K) (PROG2 (PUTPROP K
                       (CULL (PUTPROP K
                                      (FOOP (CAR (GET K
                                                      (QUOTE <VERTICES)))
                                            K)
                                      (QUOTE FOOP)))
                       (QUOTE NEIGHBORS))
              (PUTPROP K  (CULL (CDR (GET K (QUOTE FOOP))))
                       (QUOTE <VERTICES)))))
(GET SCENE (QUOTE REGIONS)))
```

SECTION THREE: FORMAT OF A SCENE.

As said before, the input to our program is a scene in a special symbolic format, which we proceed now to describe.

The Scene and its Components.- A scene is fundamentally a collection of vertices and regions.

A scene has a name which identifies it; this is an atom, with its property list containing the properties 'REGIONS', 'VERTICES', and 'BACKGROUND'. For example, the scene ONE (see figure 'ONE') has the name 'ONE'. In the property list of 'ONE' we find

REGIONS	--	(:1 :2 :3 :4 :5 :6)	Unordered list of regions composing the scene ONE.
VERTICES	--	(A B C D E F G H I J K)	Unordered list of vertices composing the scene ONE.
BACKGROUND	--	(:6)	Unordered list of regions composing the background of the scene ONE.

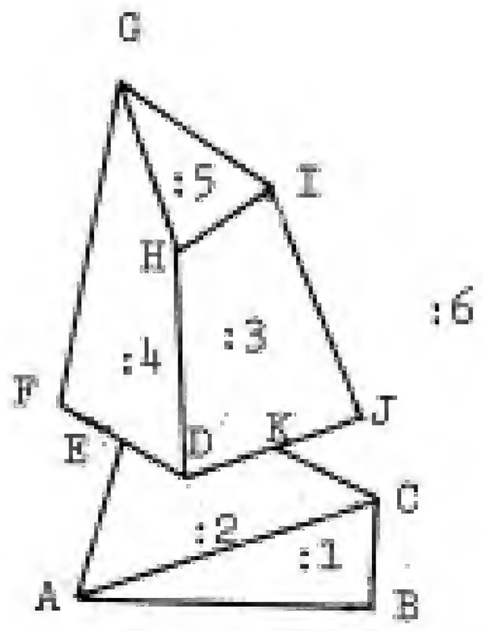


Figure 'O N E'. A scene.
Vertices and surfaces (regions)
are the main components of a scene.

Region.- A region corresponds to a surface limited by a simply connected curve. For instance, in ONE, A B C is a region, called :1, but G I J K D H is not.

Each region has as name an atom which possess additional properties describing different attributes of the region in question. These are 'NEIGHBORS', 'KVERTICES', and 'FOOP'. For example, the region in scene ONE formed by the lines AE, ED, DK, KC, C , has ':2' as name. In the property list of :2 we find:

NEIGHBORS	--	(:3 :4 :6 :1 :6)	Counterclockwise ordered list of all regions neighbors to :2. For each region, this list is unique up to cyclic permutation.
KVERTICES	--	(D E A C K)	Counterclockwise ordered list of all vertices which belong to the region :2. This list is unique up to cyclic permutation.
FOOP	--	(:3 D :4 E :6 A :1 C :6 K)	Counterclockwise ordered list of alternating neighbors and kvertices of :2. This list is unique up to cyclic permutation.

For a region (say :2), FOOP starts with any neighbor region (say :3), continues counterclockwise with the vertex belonging both to that region and that neighbor (in our example, D: it belongs to :2 and :3), continues -- counterclockwise allways -- with the next region (:4), the next vertex (E), the next region (:6), etc.

NEIGHBORS and KVERTICES are then easily derived from FOOP: take the odd-positioned elements of FOOP, and its even-positioned elements, respectively.

The FOOP property of a region is formed by a man who walks on its boundary having allways this region to his left, and takes note of the regions to his right and of the vertices (*) which finds in his way.

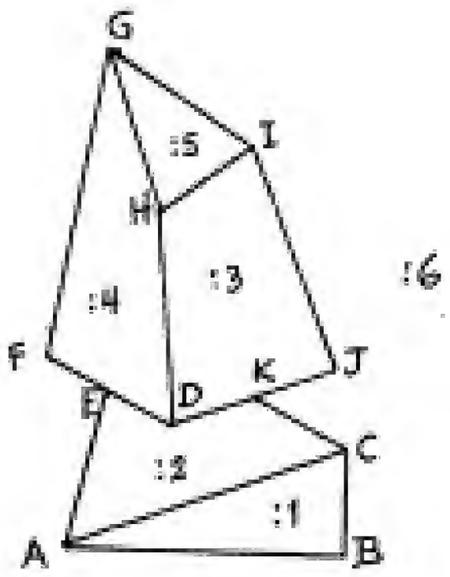


Figure 'O N E'.
Repeated here for convenience.

As another example, the property list of the region :4 (see fig. ONE) contains:

NEIGHBORS -- (:5 :6 :6 :2 :3)
KVERTICES -- (G F E D H)
FOOP -- (:5 G :6 F :6 E :2 D :3 H)

Vertex.- A vertex is the point where two or more lines of the scene meet; for instance, A, G and K are vertices of the scene ONE.

Each vertex has as name an atom which possess additional properties describing different attributes of the vertex in question. These are 'XCOR', 'YCOR', 'NVERTICES', 'NREGIONS', and 'KIND'. For example, the vertex H (see scene ONE) has in its property list:

XCOR	--	3.0	x-coordinate
YCOR	--	15.0	y-coordinate. 2-dimensional.
NVERTICES	--	(I G D)	Counterclockwise ordered list of vertices to which H is connected. (**)
NREGIONS	--	(:3 :5 :4)	Counterclockwise ordered list of regions to which H belongs (**).

(*) Places where his path changes direction or/and the region to his right changes name.

(**) This list is unique up to cyclic permutation.

KIND	--	(:3 I :5 G :4 D)	Counterclockwise ordered list of alternating nregions and nvertices of H (**).

For a vertex (say H), KIND starts with any neighbor region (say :3), continues counterclockwise (around H) with the next vertex (I in our example), then the next region (:5), etc.

The KIND property of a vertex is formed by a man who stands at the vertex, and, while rotating counterclockwise, takes note of the regions and vertices which he sees.

NREGIONS and NVERTICES are then easily derived from KIND: take the odd-positioned elements of KIND, and its even-positioned elements, respectively.

As another example, the property list of the vertex E contains:

XCOR	--	1.5
YCOR	--	3.0
NVERTICES	--	(F A D)
NREGIONS	--	(:4 :6 :2)
KIND	--	(:4 F :6 A :2 D)

(**) this list is unique up to cyclic permutation.

SECTION FOUR: TYPE OF VERTICES

Vertices are classified according to the slope, disposition and number of lines which form it. The function (TYPEGENERATOR L), where L is a list of vertices, performs this classifiication, putting in the property list of each one of the elements of L, the type to which it belongs.

The TYPE of a vertex is always a list of two elements; the first is the type-name: one of 'L', 'FORK', 'ARROW', 'T', 'K', 'X', 'PEAK', 'MULTI'; the second element is the datum; generally is a list, whose form varies with the type-name, and contains information in a determined order about the vertex in question.

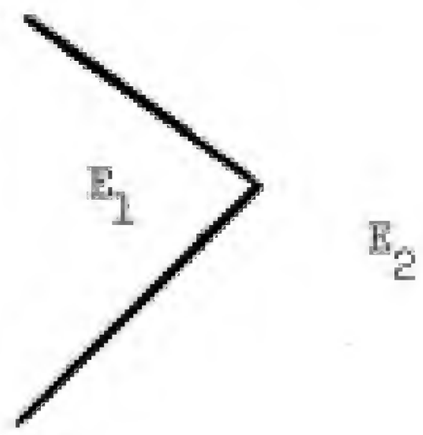
We now proceed to describe the different types, their type-names and datum.

Vertices where two lines meet.

L.- A vertex formed by only two lines is always classified as of type 'L'. Two angles exist at it, one bigger and other smaller than 180^{o}. The datum is a list of the form

$(E_1\ E_2)$, where E_1 is the region which contains the angle smaller than 180^{o}.

E_2 is the region which contains the angle greater than 180^{o}.



For instance, in scene CROSS (see fig. 'CROSS'), Q has in its property list:

TYPE -- (L (:8 :9))

The vertices of type L present in CROSS are A, F, H, I, J, O, Q.

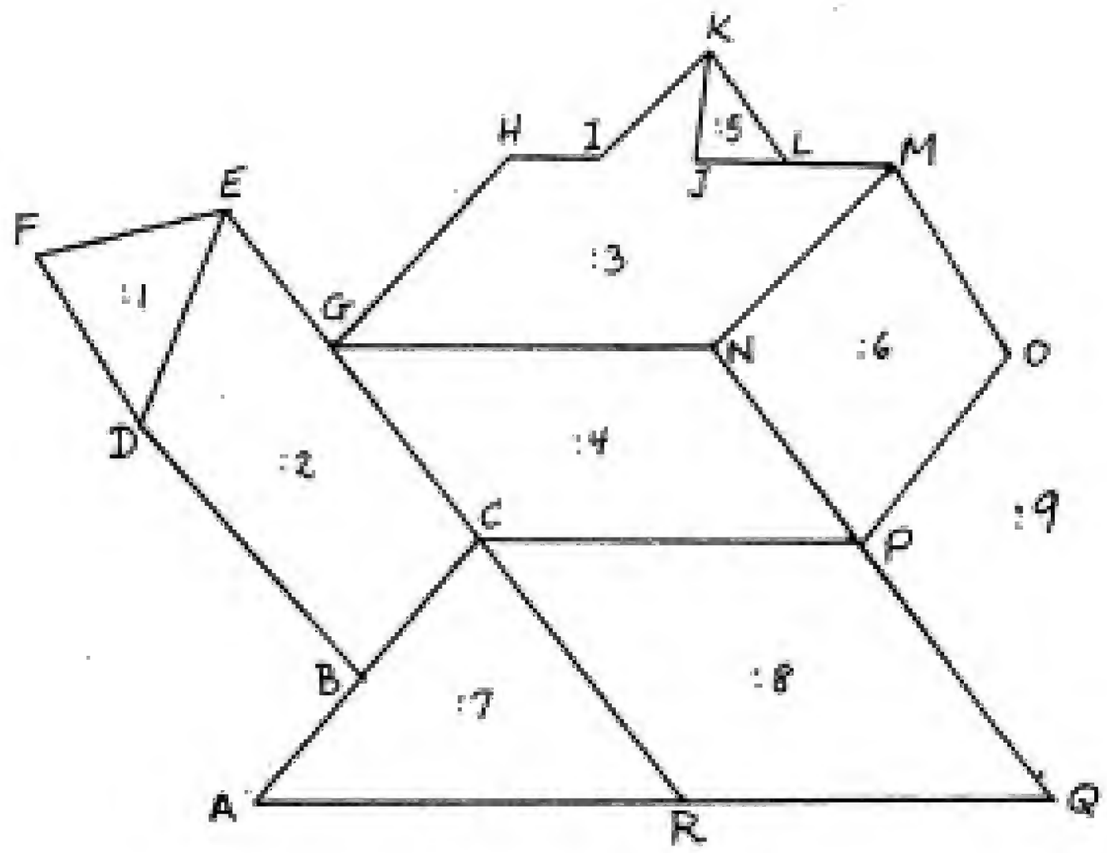


Figure 'C R O S S'.
used as an example

Vertices where three lines meet.

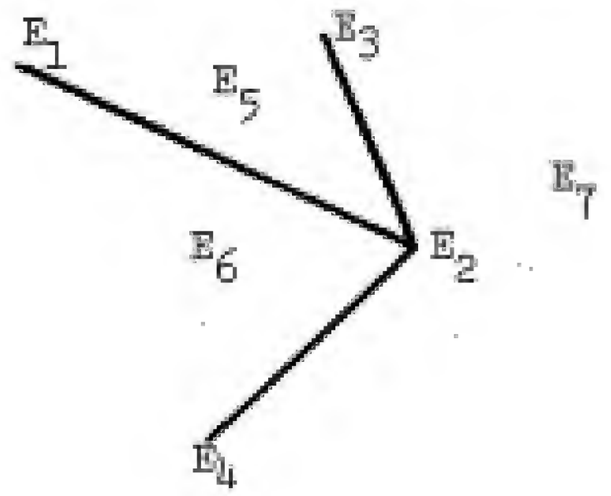
FORK.- Three lines meeting at a point and forming angles smaller than 180° form a FORK.

Its datum is the vertex itself at which the fork occurs. For instance, vertex N has in its property list

TYPE -- (FORK N)

The vertices of type FORK present in CROSS are N .

ARROW.- Three lines meeting at a point, with one of the angles bigger than 180°.



The datum of an ARROW is a list like $(E_1\ E_2\ E_3\ E_4\ E_5\ E_6\ E_7)$ where

E_1 is the vertex at the 'tail'.

E_2 is the vertex at the center.

E_3 is the vertex at the left.(*)

E_4 is the vertex at the right.

E_5 is the region at the left.

---

* With respect to segment $E_1 \rightarrow E_2$.

E_6 is the region at the right

E_7 is the region which contains the angle bigger than 180^0.

For instance, vertex E has in its property list

TYPE -- (ARROW (D E F G :1 :2 :9)) --fig. CROSS.--

The vertices of type ARROW present in CROSS are D, E, K, M.

T.- Three concurrent lines, of which two are colinear.

The datum for a T is a list of the form $(E_1\ E_2\ E_3\ E_4\ E_5\ E_6\ E_7)$, where

E_1 is the vertex at the 'tail' of the T.

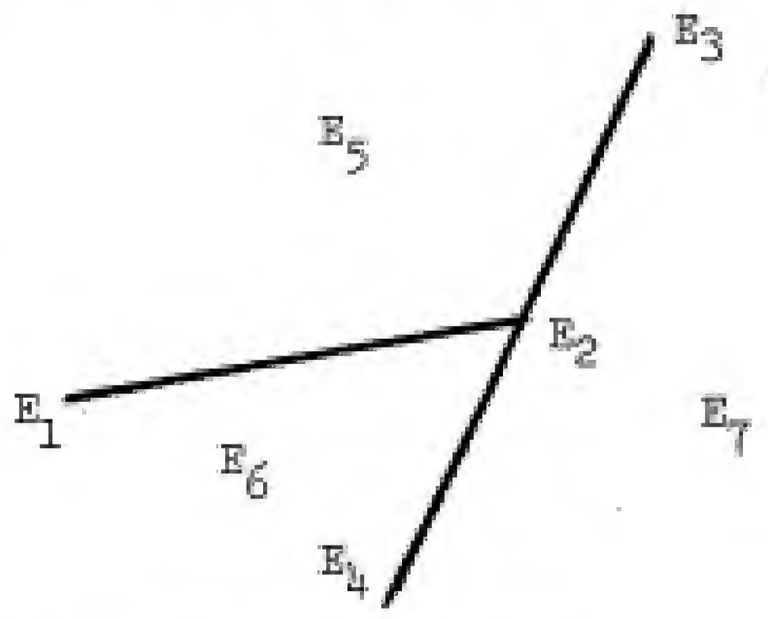
E_2 is the central vertex.

E_3 is a vertex such that $E_1\ E_2\ E_3$ is an angle between 90 and 180 degrees.

E_4 is a vertex such that $E_1\ E_2\ E_4$ is an angle smaller than 90 degrees.

That is, $E_3\ E_4\ E_2$ are colinear.

E_5 is the region which contains the angle between 90 and 180 degrees.



E_6 is the region which contains the angle smaller than 90 degrees.

E_7 is the "central" region (where the 180^0 angle is).

For instance, vertex R (fig. CROSS) has in its property list

TYPE -- (T (C R Q A :8 :7 :9))

The vertices of type T present in CROSS are B, L, R.

Vertices where four lines meet.

K.- When two of the vertices are colinear with the center, and the other two fall in the same side of such a line. The datum is a list of the form $(E_1\ E_2\ E_3\ E_4\ E_5\ E_6\ E_7\ E_8)$ where

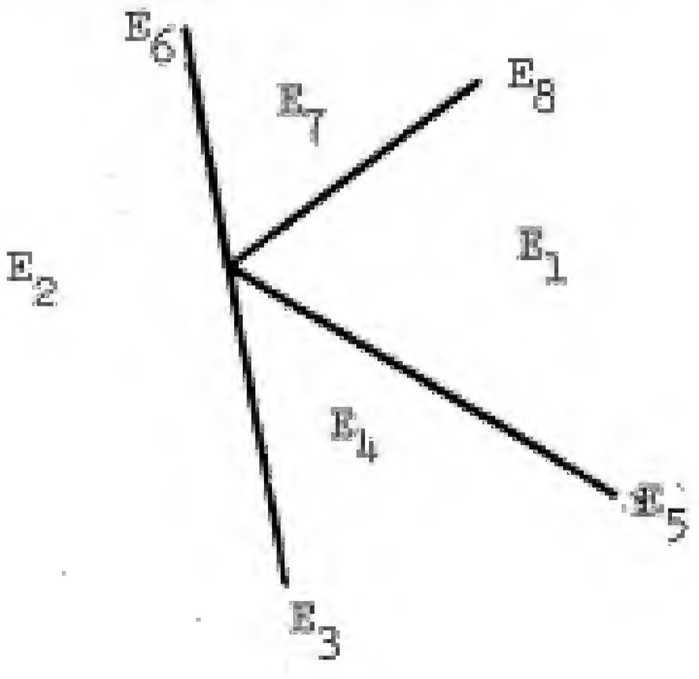
E_1 is the central region.

E_2 is the region having the 180^0 angle.

E_3 is the colinear vertex which falls to the left of $E_1\ E_2$.

E_4 the region to the left (of $E_1 \rightarrow E_2$)

E_5 the vertex to the left (of $E_1 \rightarrow E_2$)



E_6 is the colinear vertex which falls to the right of E_1 E_2.
E_7 is the region to the right of $\epsilon_1 \rightarrow \epsilon_2$.
E_8 is the other vertex to the right (of E_1).

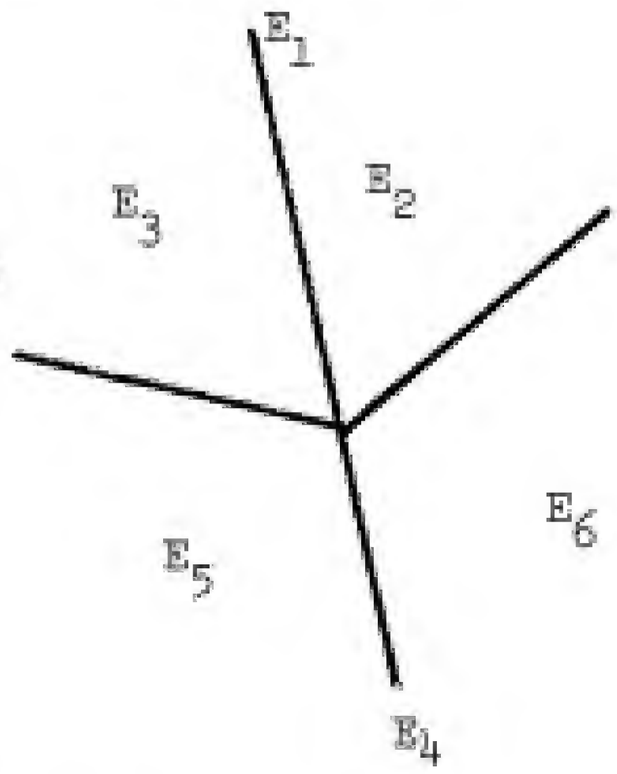
For example, we find in the property list of G the following (see fig. CROSS):

TYPE -- (K (:3 :2 C :4 N H :9 E))

G is the only vertex of type K present in scene CROSS.

X.- When two of the vertices are colinear with the center, and the other two fall in opposite sides of such a line. The datum is a list of the form (E_1 E_2 E_3 E_4 E_5 E_6), where

E_1 is one of the colinear vertices.
E_2 is the region to the left of E_1 C, where C is the vertex at the center.
E_3 is the region to the right of E_1 C,
E_4 is the other colinear vertex.
E_5 is the region to the left of E_4 C.
E_6 is the region to the right of E_4 C.



For instance, we find in the property list of P:

TYPE -- (X (N :6 :4 Q :8 :9))

The vertices of type X present in CROSS are C, P.

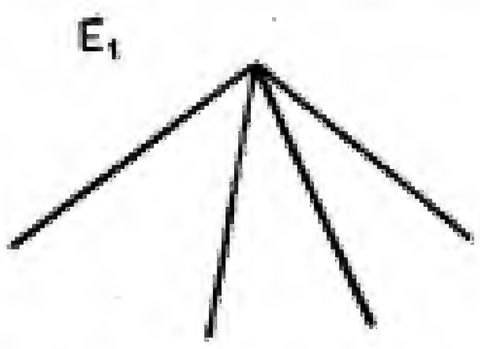
The datum for an X may also be in the form (E_4 E_5 E_6 E_1 E_2 E_3).

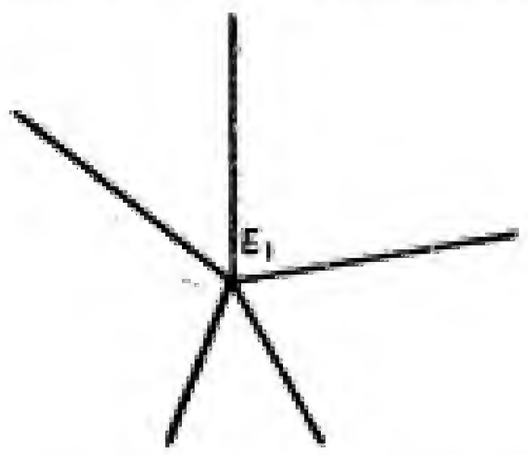
Vertices of four lines which are not of type K or X are either of type PEAK or MULTI.

Other types of vertices.

PEAK.- Formed by four or more lines, when the YCOR of the central vertex is bigger than the YCORs of the other vertices.

PEAK





MULTI

MULTI.- Vertices formed by four or more lines, and not falling in any of the

The datum for vertices of type PEAK is of the form E_1, where E_1 is the region which contains the angle bigger than 180 degrees.

The datum for vertices of type MULTI is of the form E_1, where E_1 is the vertex itself.

Here is now the function(TYPEGENERATOR L), where L is a list of vertices.

```
(DEFPROP TYPEGENERATOR (LAMBDA (A)
  (MAPC (FUNCTION
         (LAMBDA  (N)
          ((LAMBDA  (L SLOP KIND)
            ((LAMBDA  (K S)
              (PUTPROP  N
               ((LAMBDA  (K1 K2 K3 K4)
                 ((LAMBDA  (K5 K6 K7 K8)
                   (COND ((EQ (SETQ L (LENGTH KIND)) 4.)
                          (COND ((CROSSP N K2 N K4)
                                 (LIST (QUOTE L) (LIST K1 K3)))
                                (T (LIST (QUOTE L)
                                         (LIST (CAR KIND) (CADDR KIND)))
))))
                         ((EQ L 6.)
                          (COND ((EQUAL1 (CADR SLOP) (CADDDR SLOP))
                                 (LIST (QUOTE T)
                                       (LIST K6 N K4 K2 K5 K1 K3)))
                                ((EQUAL1 (CADR SLOP) (CADR S))
                                 (LIST (QUOTE T)
                                       (LIST K4 N K2 K6 K3 K5 K1)))
                                ((EQUAL1 (CADDDR SLOP) (CADR S))
                                 (LIST (QUOTE T)
                                       (LIST K2 N K6 K4 K1 K3 K5)))
                                ((EQUAL
                                  (SETQ L
                                        (LIST (CROSSP K2 N K4 N)
                                              (CROSSP K4 N K6 N)
                                              (CROSSP K6 N K2 N)))
                                  (QUOTE (T T T)))
                                 (LIST (QUOTE FORK) N))
                                ((EQUAL L (QUOTE (T T ())))
                                 (LIST (QUOTE ARROW)
                                       (LIST K4 N K2 K6 K3 K5 K1)))
                                ((EQUAL L (QUOTE (T () T)))
                                 (LIS) (QUOTE ARROW)
                                       (LIST K2 N K6 K4 K1 K3 K5)))
                                ((EQUAL L (QUOTE (() T T)))
                                 (LIST (QUOTE ARROW)
                                       (LIST K6 N K4 K2 K5 K1 K3)))
                                (T (PRINT
                                    (LIST L KIND N (QUOTE ERROR))))))
                         ((EQ L 8.)
                          (COND ((EQUAL1 (CADR SLOP) (CADDDR SLOP))
                                 (LIST (QUOTE K)
                                       (LIST K7 K3 K4 K5 K6 K2 K1 K8)))
```

```
                ((EQUAL1 (CADR SLOP) (CADR S))
                 (LIST (QUOTE X)
                       (LIST K2 K1 K3 K6 K5 K7)))
                ((EQUAL1 (CADR SLOP) (CADDDR S))
                 (LIST (QUOTE K)
                       (LIST K5 K1 K2 K3 K4 K8 K7 K6)))
                ((EQUAL1 (CADDDR SLOP) (CADR S))
                 (LIST (QUOTE K)
                       (LIST K1 K5 K6 K7 K8 K4 K3 K2)))
                ((EQUAL1 (CADDDR SLOP) (CADDDR S))
                 (LIST (QUOTE X)
                       (LIST K4 K3 K5 K8 K7 K1)))
                ((EQUAL1 (CADR S) (CADDDR S))
                 (LIST (QUOTE K)
                       (LIST K3 K7 K8 K1 K2 K5 K6 K4)))
                (T (PEAK N))))
            (CAR K)  (CADR K)  (CADDR K)  (CADDDR K)))
          (CAR KIND)  (CADR KIND)  (CADDR KIND)  (CADDDR KIND))
         (QUOTE TYPE)))
       (CDDDDR KIND)  (CDDDDR SLOP)))
     ()  (GET N (QUOTE SLOP))  (GET N (QUOTE KIND)))))
  A))
EXPR)

(DEFPROP PEAK (LAMBDA (N)
  (PROG (B R A C)
        (SETQ A (GET N (QUOTE KIND)))
        (SETQ B (GET N (QUOTE YCOR)))
        (SETQ C (CAR (LAST A)))
        D
        (COND ((NULL A) (RETURN (LIST (QUOTE PEAK) R)))
              ((LESSP B (GET (CADR A) (QUOTE YCOR)))
               (RETURN (LIST (QUOTE MULTI) N)))
              ((CROSSP C N (SETQ C (CADR A)) N) (GO E))
              (T (SETQ R (CAR A))))
        E
        (SETQ A (CDDR A))
        (GO D)))
EXPR)

(DEFPROP SLOPGENERATOR (LAMBDA (A)
  (MAPC (FUNCTION
         (LAMBDA (N)
                 (PUTPROP  N
                  (MAPCAR (FUNCTION
                           (LAMBDA (V)
                                   (COND ((GET V (QUOTE XCOR))
                                          (SLOP N V))
                                         (T V))))
                          (GET N (QUOTE KIND)))
                  (QUOTE SLOP))))
         A))
EXPR)
```

The function SLOPGENERATOR generates the indicator SLOP, for each vertex.

SECTION FIVE: THE PROGRAM

As already mentioned, the recognition program called SEE accepts a description of a scene expressed in the notation just described, and produces as output lists containing the identification of the bodies present in the scene.

In this section we describe the program, and how it achieves its goals. We begin with a couple of examples

Example 1. Scene 'II'.- This scene (see figure 'II') is analyzed by SEE, with the following results:

```
(LOCAL ASSUMES  (:5)  (:13 :14)  SAME BODY)
     (BODY 1. IS :1 :3 :2 :18 :17)
     (BODY 2. IS :4 :16 :15)
     (BODY 3. IS :7 :6 :11 :12)
     (BODY 4. IS :9 :8 :10)
     (BODY 5. IS :13 :14 :5)
     (BODY 6. IS :20 :19)
```

Results for scene 'II' (see next page)

Example 2. Scene 'BRIDGE'.- With this example, we give the details of the operation. We start by loading into LISP the following files (currently, from microtape GUZMAN *):

(UREAD SQRT LAP2) ↑Q	Square root, defined in LAP
(REMLAP T)	Kill LAP
(UREAD SEE 25) ↑Q	Read SEE
(UREAD GENERA 10) ↑Q	Read auxiliary functions
(UREAD BRIDGE SCEN3) ↑Q	Loading of this file causes scene BRIDGE to be read (in the format described in Section Two) and transformed to the proper form which SEE expects (as described in Section Three).

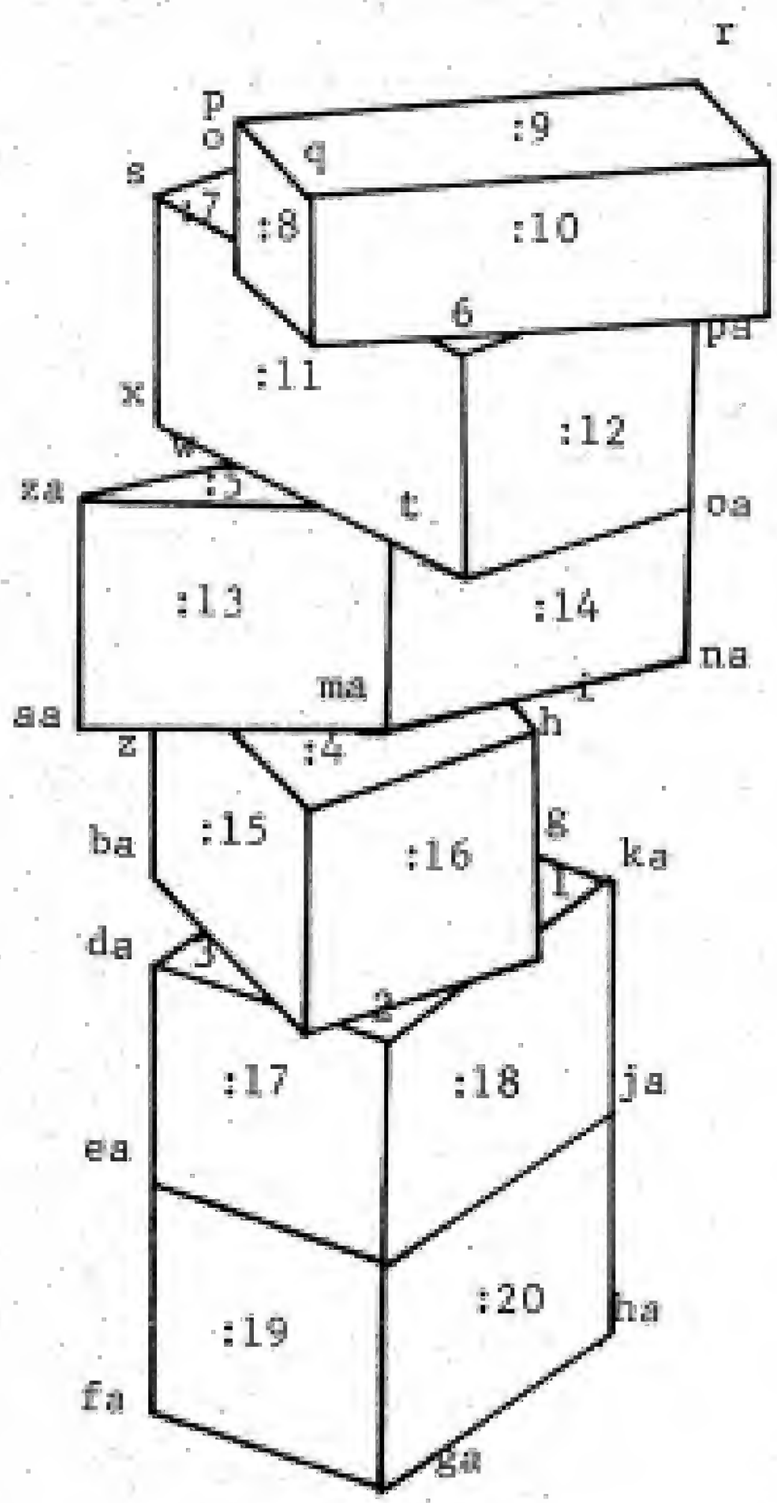


Figure 'I I'.
This scene is analyzed by SEE, with results detailed in Example 1.
All bodies are correctly found.
Some of the vertices appear in the drawing with their names; in other drawings we will generally omit them; we are also omiting the coordinate axes.

```
          (SEE (QUOTE BRIDGE))
```

We call to SEE, with the name of the scene.

Different parts of the program print their name as they are entered.

```
SEE 25 ANALYZES BRIDGE
FILLSCENE
CLEAN
EVIDENCE
LOCALEVIDENCE
GLOBAL
((NIL) ((:8) G0017 G0015 G0014 G0013 G0012 G0011) ((:7) G0022) ((:10) G0023 etc.
LOCAL
(LOCAL ASSUMES (:18) (:19) SAME BODY)
(LOCAL ASSUMES (:28) (:29) SAME BODY)
(LOCAL ASSUMES (:10) (:8 :11 :5 :6 :4) SAME BODY)
(LOCAL ASSUMES (:7) (:8 :11 :5 :6 :4 :10) SAME BODY)
((NIL) ((:8 :11 :5 :6 :4 :10 :7) G0013 G0014 G0011 G0017 G0012 G0013 G0020 etc.
LOCAL
(SINGLEBODY ASSUMES (:19 :18) (:16) SAME BODY)
(((:24 :9 :21 :27 :12 :25) G0013 G0060 G0056 G0054 G0049 G0033 G0029 G0057 etc.
LOCAL
RESULTS
(BODY 1. IS :24 :9 :21 :27 :12 :25)
(BODY 2. IS :22 :26 :23)
(BODY 3. IS :17 :3 :20)
(BODY 4. IS :1 :2)
(BODY 5. IS :14 :15 :13)
(BODY 6. IS :19 :18 :16)
(BODY 7. IS :29 :28)
(BODY 8. IS :8 :11 :5 :6 :4 :10 :7)
NIL
```

Results for scene 'BRIDGE'

Unnecessary detail has been removed from the drawings; the reader must bear in mind that the complete scene contains vertices (whose names do not appear in the drawings) and their corrdinates with respect to axis which are also not shown, lines --edges-- and surfaces --regions--.

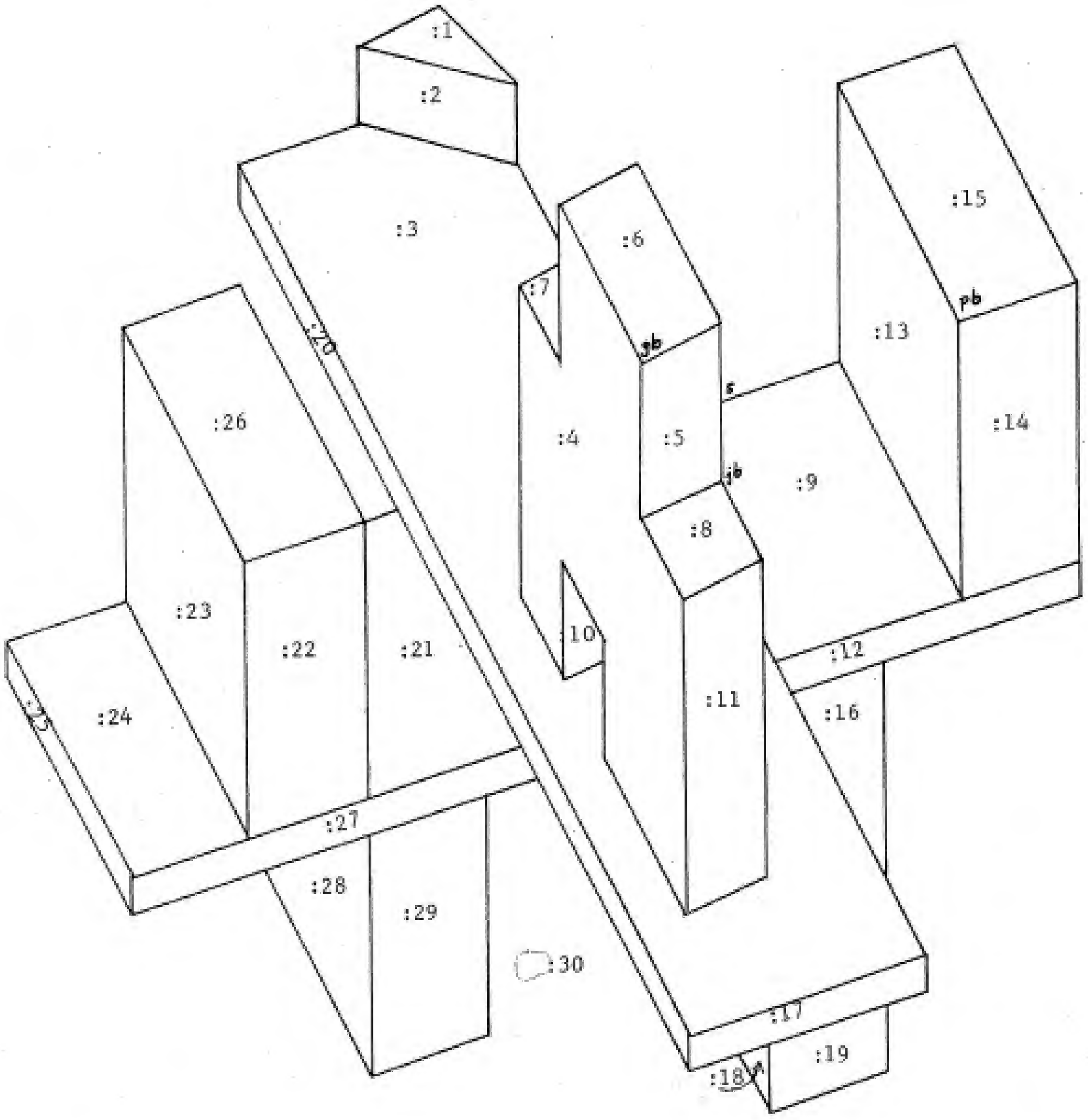


Figure 'B R I D G E'.
The long body :25 :24 :27 :21 :9 :12 is correctly identified.

How the program works.- The operation of SEE is quite straighforward; the program is little recursive, and does not do any search. Its main parts, which are executed one after another, unless otherwise indicated, are:

FILLSCENE.- The properties SLOP and TYPE are generated, for each vertex, if they were not already present.

CLEAN.- Old properties used internally by SEE are removed from vertices and regions.

EVIDENCE.- An analysis is made of vertices, regions and associated information, in search for clues that indicate that two regions form part of the same body. If evidence exists that two regions in fact belong to the same body, they are linked or marked with a genaym ("both receive a label of the same color"). These links may be strong (global) or weak (local).

LOCALEVIDENCE.- Some features of the scene will weakly suggest that some group of regions should be considered together, as part of the same body. This part of the program puts these 'local' links or evidences.

GLOBAL.- The 'strong' links gathered so far are analyzed; regions are grouped into "nuclei " of bodies, which grow until some conditions fail to be satisfied (a detailed explanation follows later).

LOCAL.- Weak evidence is taken into account for deciding which of the unsatisfactory global links should be considered satisfactory, and the corresponding nuclei of bodies are then joined, to form a single and bigger nuclei . Assumptions made by LOCAL are printed (see output of SEE). LOCAL may call GLOBAL again, or go on.

SINGLEBODIES.- If a single region does not belong to a bigger nucleia, but is linked by one strong evidence to other region, it is incorporated into the nuclei of the last one. This part of the program may call GLOBAL or LOCAL again, if

necessary, or go on. 'SINGLEBODIES' also prints its assumptions.

RESULTS.- The regions belonging to the background are screened out, and the results are printed.

Example. Scene TOWER.- At this point I want to go into considerable detail to explain each one one of the parts of SEE that have been sketched above. This section is important because it will help to understand the behavior of SEE, its strength and defficiences.

First, we begin by showing a typical analysis of SEE with a somewhat complicated scene (see fig. 'TOWER').

The output is

```
 SEE 25  ANALYZES TOWER
CLEAN
EVIDENCE
LOCALEVIDENCE
GLOBAL
((NIL) ((:20) G0055 G0054 G0053 G0052) ((:19) G0059 G0056  etc.
LOCAL
((NIL) (:20 :19 :21) G0055 G0052 G0059 G0056 G0054 G0053) ((: etc.
LOCAL
(((:3 :2 :1) G0077 G0089 G0079 G0078 G0076) ((:5 :15 :4) G0049  etc
LOCAL
RESULTS
(BODY 1. IS :3 :2 :1)
(BODY 2. IS :5 :15 :4)
(BODY 3. IS :23 :17)
(BODY 4. IS :6 :7 :8)
(BODY 5. IS :10 :11 :9)
(BODY 6. IS :13 :14 :12)
(BODY 7. IS :18 :22)
(BODY 8. IS :20 :19 :21)
NIL
```

Results for TOWER (see next page)

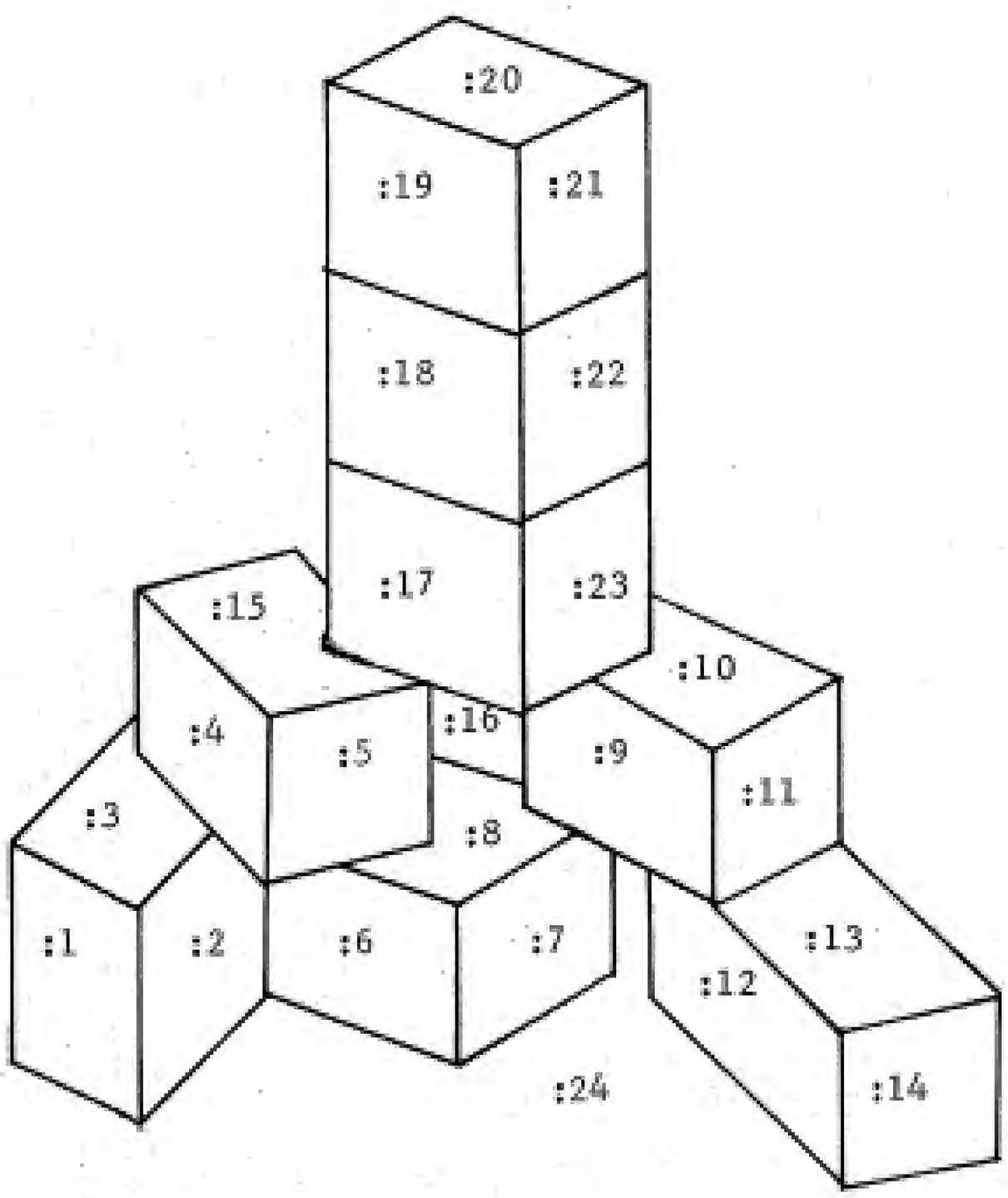


Figure 'T O W E R'.
Neither LOCAL or SINGLEBODIES are necessary to correctly parse this scene into the bodies which form it. There are plenty of 'strong' links in this scene, and SEE makes good use of them.

FILLSCENE, CLEAN.- These two parts of SEE are simple; if necessary, FILLSCENE calls SLOPGENERATOR and TYPEGENERATOR; CLEAN removes some unwanted properties. The beginning of the definition of SEE thus looks like

```
(DEFPROP SEE (LAMBDA (S)
  (PROG (VERTICES  REGIONS  SLOP  TYPE  BODIES  U  LEV  BACKGROUND  SMB
         Z  SW1  W)
        (PRINT (QU (SEE 25 ANALYZES (EV S))))
        (SETQ SLOP (QUOTE SLOP))  (SETQ TYPE (QUOTE TYPE))
        (SETQ VERTICES (GET S (QUOTE VERTICES)))
        (SETQ BACKGROUND (GET SCENE (QUOTE BACKGROUND)))
        (COND ((NULL (SETQ REGIONS (GET S (QUOTE REGIONS))))
               (RETURN ()))
              (CLEAR (GO FILLSCENE))  (T (GO CLEAN)))
      FILLSCENE  (PRINT (QUOTE FILLSCENE))  (SLOPGENERATOR VERTICES)
        (TYPEGENERATOR VERTICES)  (SETQ CLEAR ())  CLEAN
        (PRINT (QUOTE CLEAN))  (CLEAN VERTICES (QUOTE NEXTE))
        (REMPROP S (QUOTE BODIES))  (CLEAN REGIONS (QUOTE BODY*))
        (CLEAN REGIONS (QUOTE BODY))
        (MAPC (FUNCTION (LAMBDA (J) (REMPROP (CADR J) (QUOTE FAS))))
              TES)
        (SETQ TES ())
```

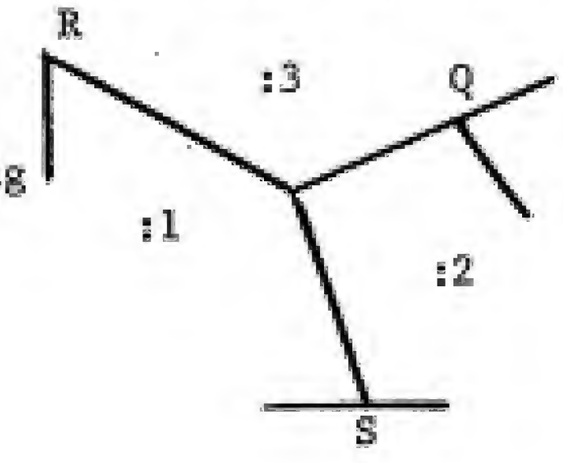
EVIDENCE.- Strong or global links are placed by this part of the program. Two functions are used: EVERTICES and TJOINTS.

```
EVIDENCE  (PRINT (QUOTE EVIDENCE))
(MAPC (FUNCTION EVERTICES (SECTN
(MAPC (FUNCTION TJOINTS) TES)
```

EVERTICES.- This function consideres each vertex of the scene, under the following rules:

L.- Vertices of the type 'L' are not considered.

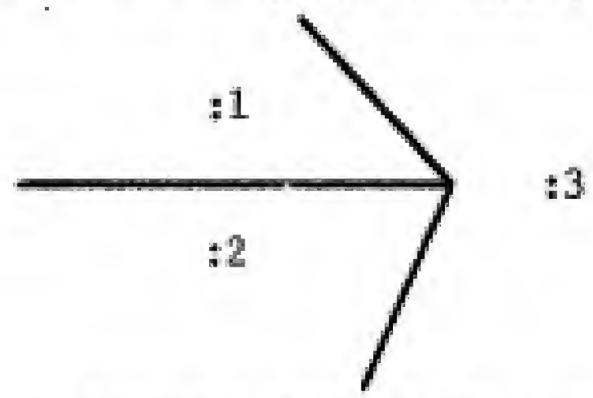
FORK.- If three regions meet at a vertex of the FORK type, and none of them is the background, links between them will be formed. For instance, in fig. TOWER we establish the links :19-:20, :19-:21, :20-:21. Neverthe less, some of these links may not be produced: in the figure at the right, the link between :3 and :2 is not produced because Q is a "passing T"; the link between :1 and :2 <u>is</u> produced. The link between :1 and :3 is <u>not</u> generated because R is an 'L'.



This is a powerful heuristic, and (see fig. 'BRIDGE') allows us, for instance, to omit the link between regions :5 and :9 in scene 'BRIDGE'. Nevertheless, this same heuristic puts a link among regions :8 and :9 of the same scene. As we shall see later, this is not too bad.

ARROW.- This type of vertex causes two of its regions (the 'left' and the 'right' one) to be joined; In the figure of the right, we will put a link between :1 and :2, which counts as evidence that :1 and :2 belong to the same body. Nothing is said about :3.

For instance, is this type of vertex that joints :1 and :2 in figure 'BRIDGE'.

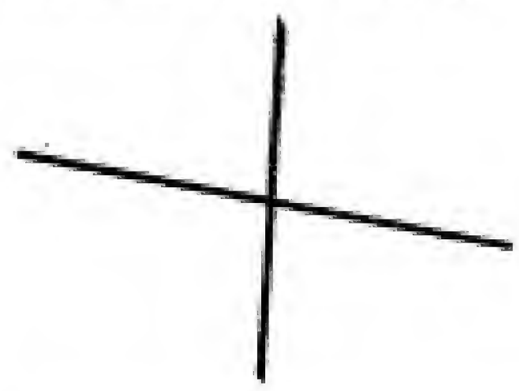


Arrow links :1 and :2

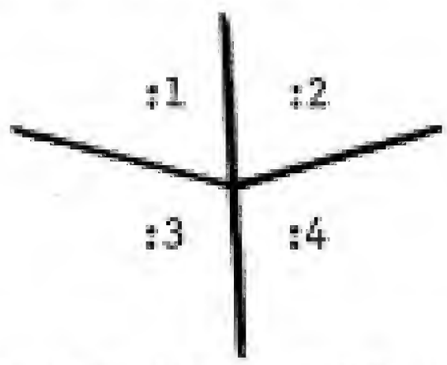
X.- Two cases are distinguished:

a) The X is formed by the intersection of two lines. No evidence is formed.

b) Otherwise, links :1-:2 and :3-:4 are formed (see fig. at the right). This second type of X occurs when we have piles of objects; for instance, in fig. TOWER, :18 and :22 are considered as belonging to the same body, due to this type of vertex.



X: No links established.



X: two links established.

PEAK.- All its regions, except the one containing the obtuse angle, are linked to each other. See fig. CORN.

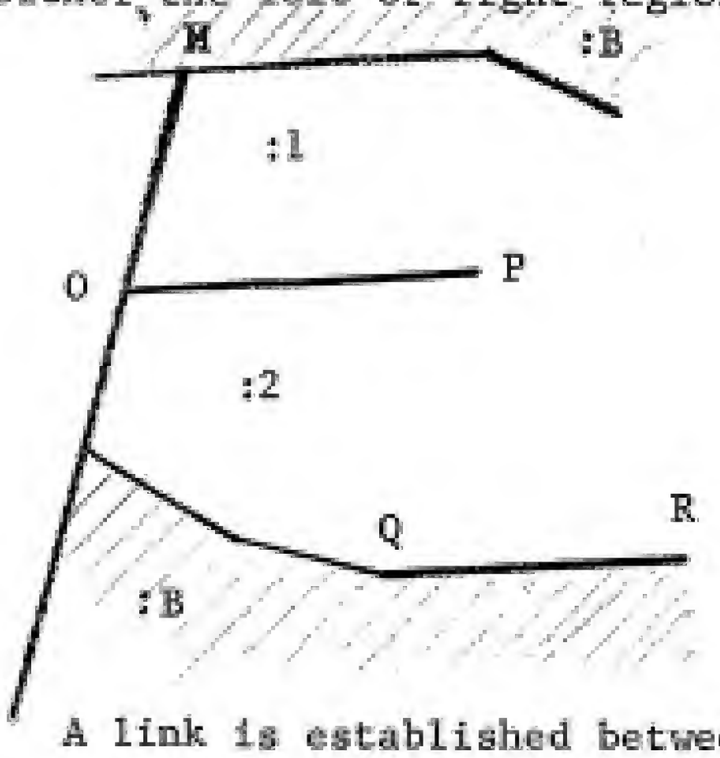
T.- We first look for another T to match the vertex currently analyzed; two T's match if they are colinear and "facing each other"; the closest pair is chosen, if several. For instance, A and B are paired (fig. at the right). An indicator 'NEXTE' is placed in such vertices.

Matching Ts.

```
(DEFPROP EVERTICES (LAMBDA (V)
  ((LAMBDA  (TYPE NREGIONS BACKGROUND)
    (COND ((EQ (CAR TYPE) (QUOTE FORK))
           (SETQ TYPE (GET V (QUOTE KIND)))
           (COND ((AND (NULL (GET (CAR NREGIONS) BACKGROUND))
                       (NULL (GET (CADR NREGIONS) BACKGROUND))
                       (NULL (GET (CADDR NREGIONS) BACKGROUND)))
                  ((LAMBDA  (K3 K4 K5 K6)
                    (AND (OR (TPASSCOR V (CADR TYPE))
                             (GEV (CAR TYPE) K3))
                         (OR (TPASSCOR V K4) (GEV K3 K5))
                         (OR (TPASSCOR V K6) (GEV (CAR TYPE) K5))))
                   (CAR (SETQ ** (CDDR TYPE)))
                   (CAR (SETQ ** (CDR **)))  (CAR (SETQ ** (CDR **)))
                   (CADR **)))))
          ((EQ (CAR TYPE) (QUOTE ARROW))
           (SETQ U (CDDDDR (CADR TYPE)))  (GEV (CAR U) (CADR J)))
          ((EQ (CAR TYPE) (QUOTE X))
           (SETQ U (CADR TYPE))
           (COND ((COLINEAL
                   (CAR (SETQ U
                              (REMOVE  (CAR U)
                               (REMOVE  (CADDDR U)
                                (GET V (QUOTE NVERTICES))))))
                  V  (CADR U)))
                 (T (SETQ U (CDADR TYPE))  (GEV (CAR J) (CADR J))
                    (SETQ U (CDDDR U))  (GEV (CAR U) (CADR U))))))
          ((EQ (CAR TYPE) (QUOTE PEAK))
           (MAP (FUNCTION
                 (LAMBDA  (J)
                  (MAPC (FUNCTION (LAMBDA (K) (GEV (CAR J) K)))
                        (CDR J))))
                (REMOVE (CADR TYPE) NREGIONS)))
          ((EQ (CAR TYPE) (QUOTE T))
           (SETQ U (CDDDDR (CADR TYPE)))
           (COND ((AND (NULL (GET (CAR J) BACKGROUND))
                       (NULL (GET (CADR U) BACKGROUND))
                       (HAVING
                        (FUNCTION (LAMBDA (X) (GET X BACKGROUND)))
                        (GET (CAR U) (QUOTE NEIGHBORS)))
                       (HAVING
                        (FUNCTION (LAMBDA (X) (GET X BACKGROUND)))
                        (GET (CADR U)(QUOTE NEIGHBORS)))
                       ((LAMBDA  (A S)
                         (AND (PBACK (PROG2 ()  A
                                            (SETQ A (CAR (LAST A))))
                                     (PBACK (PROG2 ()
                                                   (SETQ A
                                                         (GET (CADR J)
                                                              (QUOTE FOOP))))
                                            (SETQ A (CAR (LAST A))))))
                        (GET (CAR U)(QUOTE FOOP))
                        (SLOP (CAADR TYPE) V)))
                  (GEV (CAR U)(CADR U))))
                 (THROW (SLOP (CAADR TYPE) V) (CADR TYPE)))))))
  (GET V (QUOTE TYPE))  (GET V (QUOTE NREGIONS))  (QUOTE BACKGROUND)))
EXPR)
```

a) Once the NEXTE is determined, we establish a link between :1 and :2 --last figure-- and another between :3 and :4. These links will not be produced if the result of them is to associate the background with something which is not the background.

b) The following test is done: If neither the left or right region (:1 or :2 in figure at right) is the background, but they have it as a neighbor, and in some part of the boundary of :1 and :2 with the background, the segment which separates them is parallel to the central segment of the T, then :1 and :2 are linked.



A link is established between :1 and :2 because they do not belong to the background, but this is a neighbor of both of them, and the segment that separates :1 of the background (and the same is true for :2) is parallel to OP, the central segment of the T in question. (B is the background).

For instance, in figure 'I I', this analysis applied to the vertex T will produce evidence that :13 and :14 belong to the same body (ZA - AA, T - MA and OA - NA are parallel). This is a rather global heuristic although only useful for bodies with parallel faces. Also, EVERTICES classifies T's according to the slope of their central segments, storing them in the form

TES = ((3.5 G0001) (-10.666 G0002) ...) The numbers are slopes, and the gensyms contain under property 'FAS' a list of datums of Ts having that slope.

TJOINTS.- This function actuates on TES and establishes global evidences as described in part a) of T. (top of this page).

LOCALEVIDENCE.- A leg is an arrow if one of its left or right vertices is a corner (if necessary, through a chain of matched Tes) which has a side parallel to the central segment of the arrow. The function LEGS finds legs in the scene, and stores this information in a list of 'weak' links.

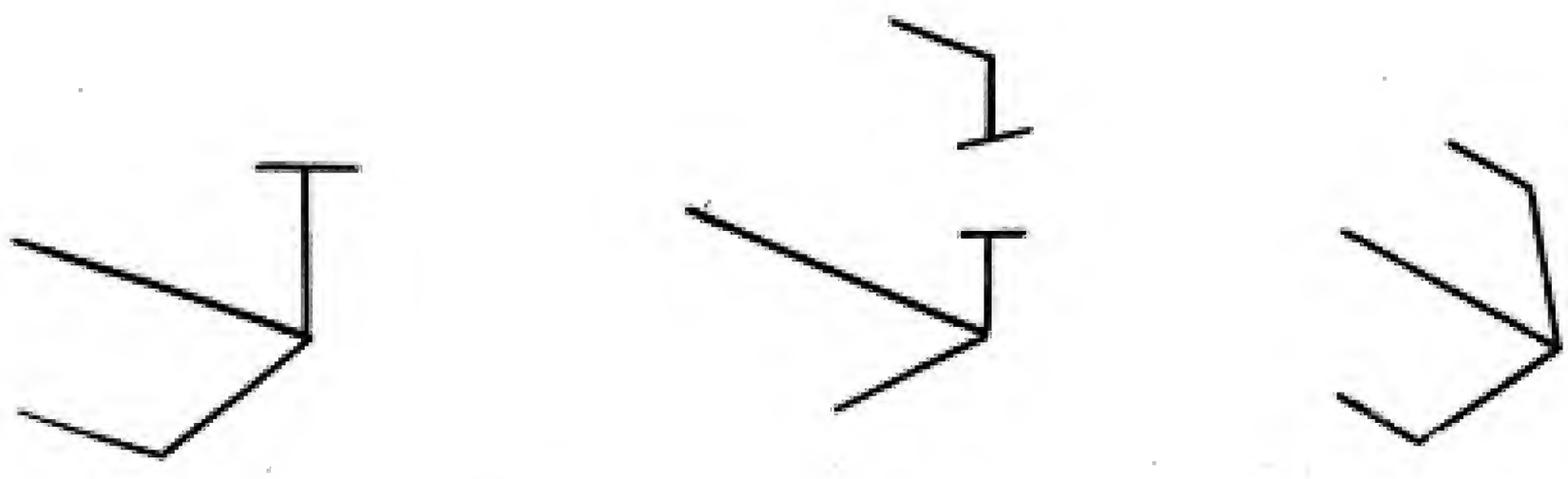


Figure 'L E G S'.
Three different types of legs.

The definition of the function (LEGS N), where N is a vertex, follows.

```
(DEFPROP LEGS (LAMBDA (N)
  (AND (EQ (CAR (SETQ U (GET N (QUOTE TYPE)))) (QUOTE ARROW))
       (SETQ U (CADR U))
       (OR (AND (SETQ N (CORNER (CADR U) (CADDR U)))
                (OR (PARALLEL  (CADR U)  (CAR U)  N
                     (CAR(GET N (QUOTE NVERTICES))))
                    (PARALLEL  (CADR U)  (CAR U)  N
                     (CADR (GET N (QUOTE NVERTICES))))))
           (AND (SETQ N (CORNER(CADR U) (CADDDR U)))
                (OR (PARALLEL  (CADR U)  (CAR U)  N
                     (CAR (GET N (QUOTE NVERTICES))))
                    (PARALLEL  (CADR U)  (CAR U)  N
                     (CADR (GET N (QUOTE NVERTICES))))))
       (SETQ U (CDDDDR U))  (LEV (CAR U) (CADR U))))
  EXPR)
```

```
(DEFPROP CORNER (LAMBDA (A B)
  (COND ((EQ (CAR (SETQ ** (GET B (QUOTE TYPE)))) (QUOTE L)) B)
        ((NOT (EQ (CAR **) (QUOTE T))) NIL)
        ((EQ (CAR (SETQ ** (CADR **))) A)
         (COND ((SETQ ** (GET B (QUOTE NEXTE)))
                (CORNER ** (CAADR (GET ** (QUOTE TYPE))))))
        ((EQ (CADDDR **) A) (CORNER B (CADDR **)))
        ((EQ (CADDR **) A) (CORNER B (CADDDR **)))))
  EXPR)
```

GLOBAL. - Strong evidence is analyzed in this part of the program. When this section is entered, several links (strong and weak) exist among the different regions of the scene. These links are shown pictorially in figure 'LINKS', for the scene 'BRIDGE' (see both). All the links to the background (:30) are deleted: the background can not be part of any body.

Definition: a nucleus (of a body) is either a region or a set of nuclei which has been formed by the following rule.

Rule: If two nuclei are connected by two or more links, they are merged into a larger nucleus by concatenation.

(Two nuclei A and B are linked if regions a and b are linked where acA and bcB). For instance, regions :8 and :11 are put together, because there exist two links among them, to form the nucleus :8 - 11. Now, we see that region :4 (see fig. 'LINKS') has two links with this nucleus :8 - 11, and therefore the new nucleus :8 - 11 - 14 is formed.

We let the nuclei grow and merge under the former rule, until no new nuclei can be formed. When this is the case, the scene has been partitioned into several "maximal" nuclei ; between any two of these, there are zero or, at most, one link. For example, fig. 'LINKS' will be transformed into fig. 'NUCLEI'.

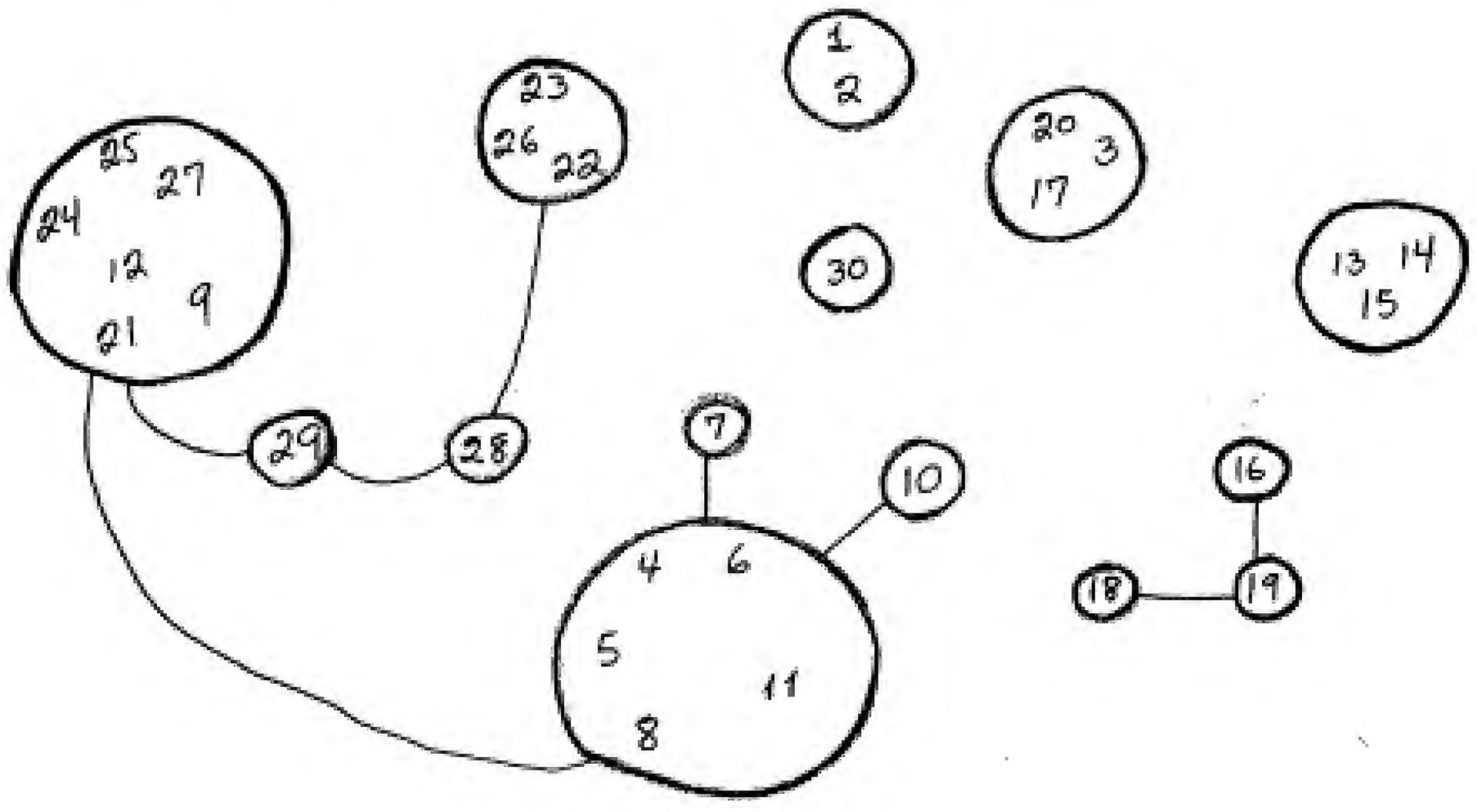


Figure 'N U C L E I '
(See fig. 'LINKS' first
on next page).

After joining all nuclei having two or more links in common, the representation for the scene 'BRIDGE' changes from that

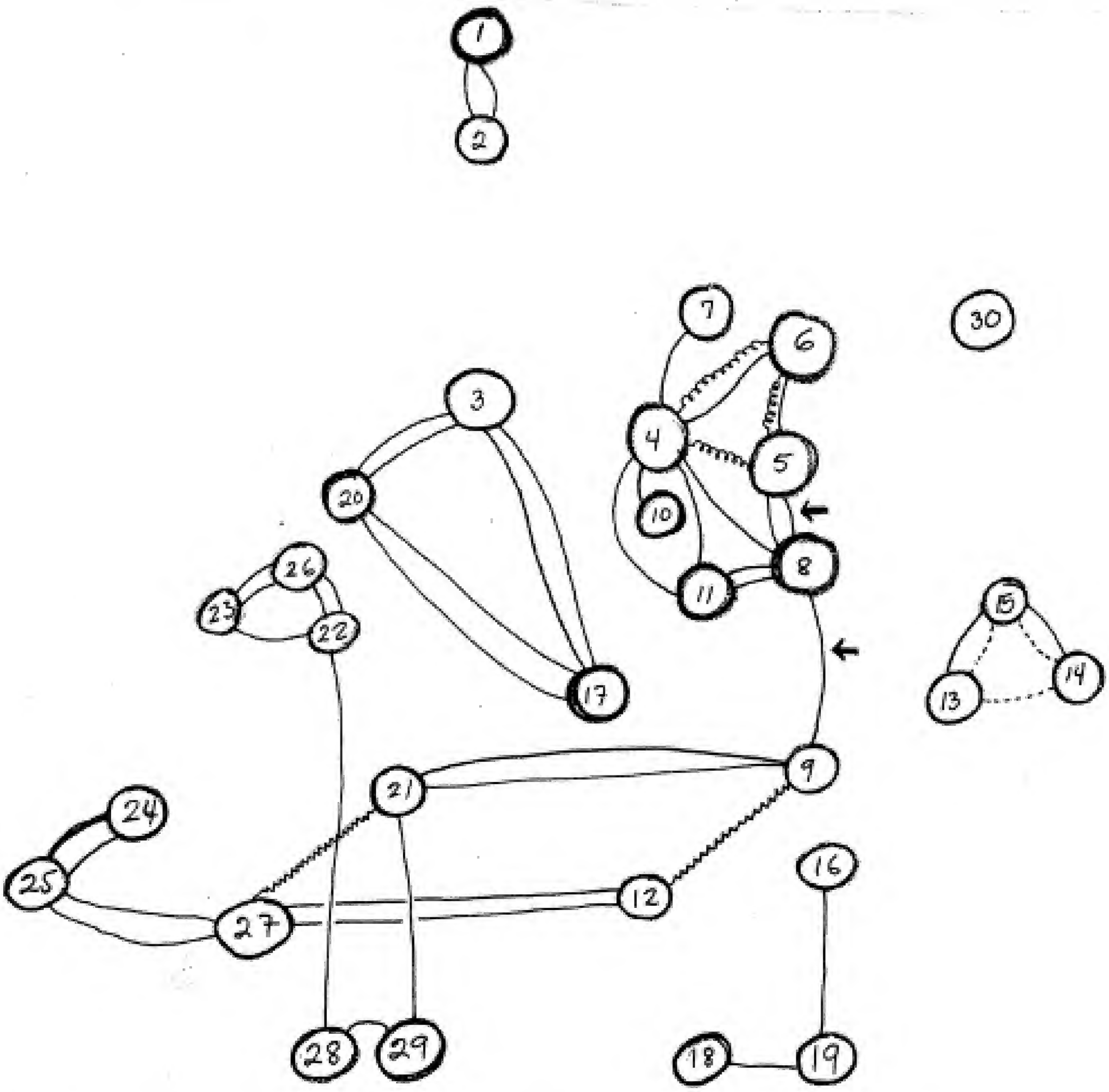


Figure 'L I N K S'

This figure represents scene 'BRIDGE', with the strong links between its regions (represented here by circles) shown. The dotted links represent the evidence generated by the vertex PB (see BRIDGE). The short arrows show the links put by vertex JB; note that a link between :5 and :9 is not put, because S (see fig. BRIDGE) is a passing t-joint. Zig-zag links are produced by the mechanism described in part b) of T. (page 31).
Curled links are produced by vertex GB; even if this vertex were occluded, and the links missing, there is still enough evidence to identify regions :4 :5 and :6 as belonging to the same body. Weak links are not shown. See text.

LOCAL. - If some (strong) link joining two "maximal" nuclei is also reinforced by a weak link, these nuclei are merged.

For example, in scene BRIDGE (see fig. 'BRIDGE') the following local links exist (among others): :7 to :4; :10 to :4; :28 to :29; :18 to :19. Therefore, the corresponding nuclei are merged and now fig. NUCLEI looks like:

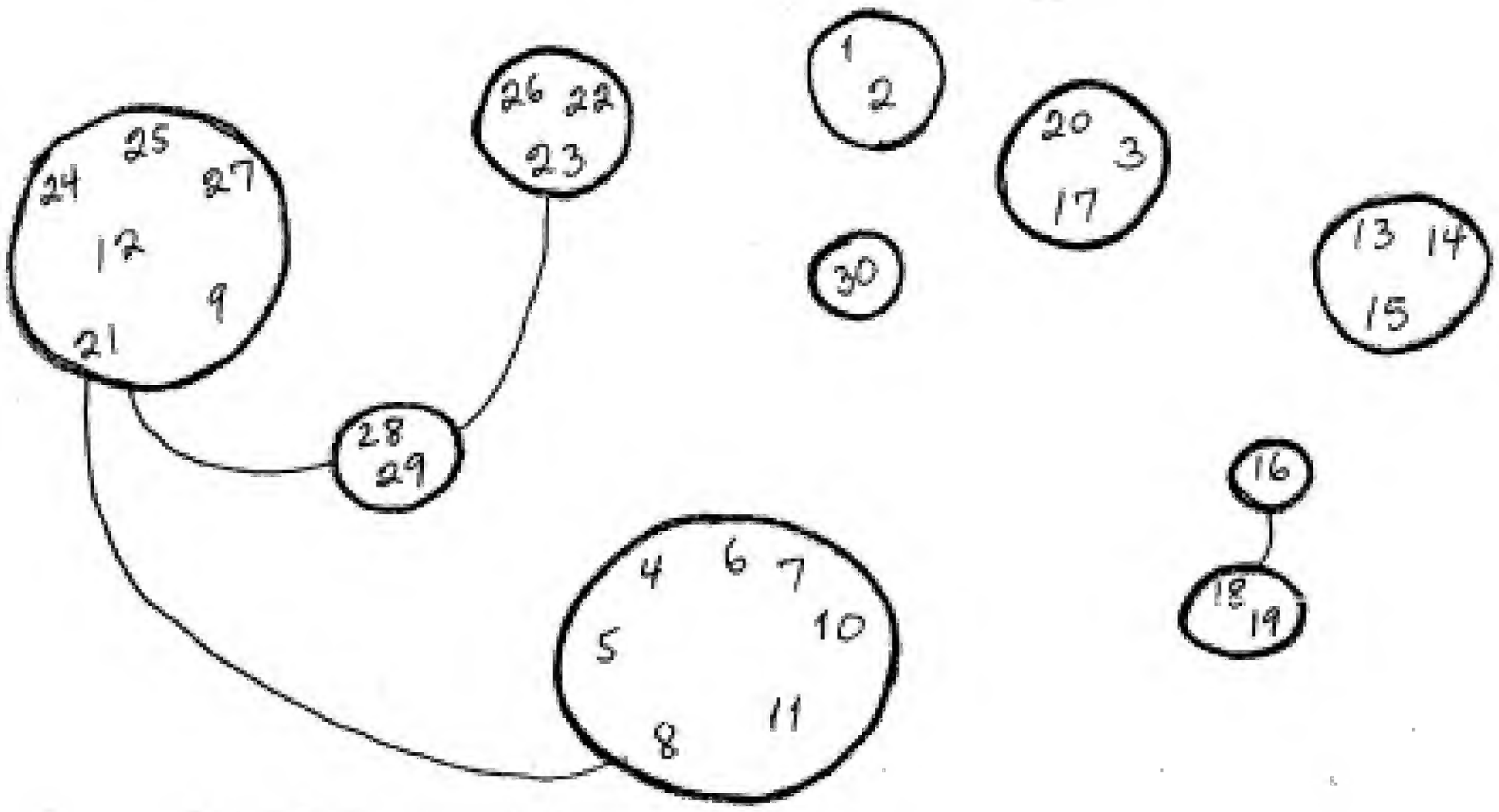


Fig. 'N E W N U C L E I '

The figure shows the scene 'BRIDGE', after LOCAL transforms it from the representation in fig. 'NUCLEI ' to the one shown here.

A weak link does not cause the regions which it links to be merged or considered as belonging to the same body, unless there is in addition one strong evidence among such regions. LOCAL may call GLOBAL again, if necessary.

SINGLEBODIES. - A strong link joining a nucleus and another nucleus composed by a single region is considered enough evidence and the nuclei in question merged, if there is no other link emanating of such single region. For instance, in fig. 'NEW NUCLEI ', nucleus :16 is merged with nucleus :18 - 19 (see fig. 'FINAL'). Nucleus :28 -29 is not joined with :26 - 22 - 23 or with :24 - 25 - 27 - 12 - 21 - 9. Even if nucleus :28 - 29 were composed by a single region, still will not be merged, since two links emerge from it: two nuclei claim its possession.

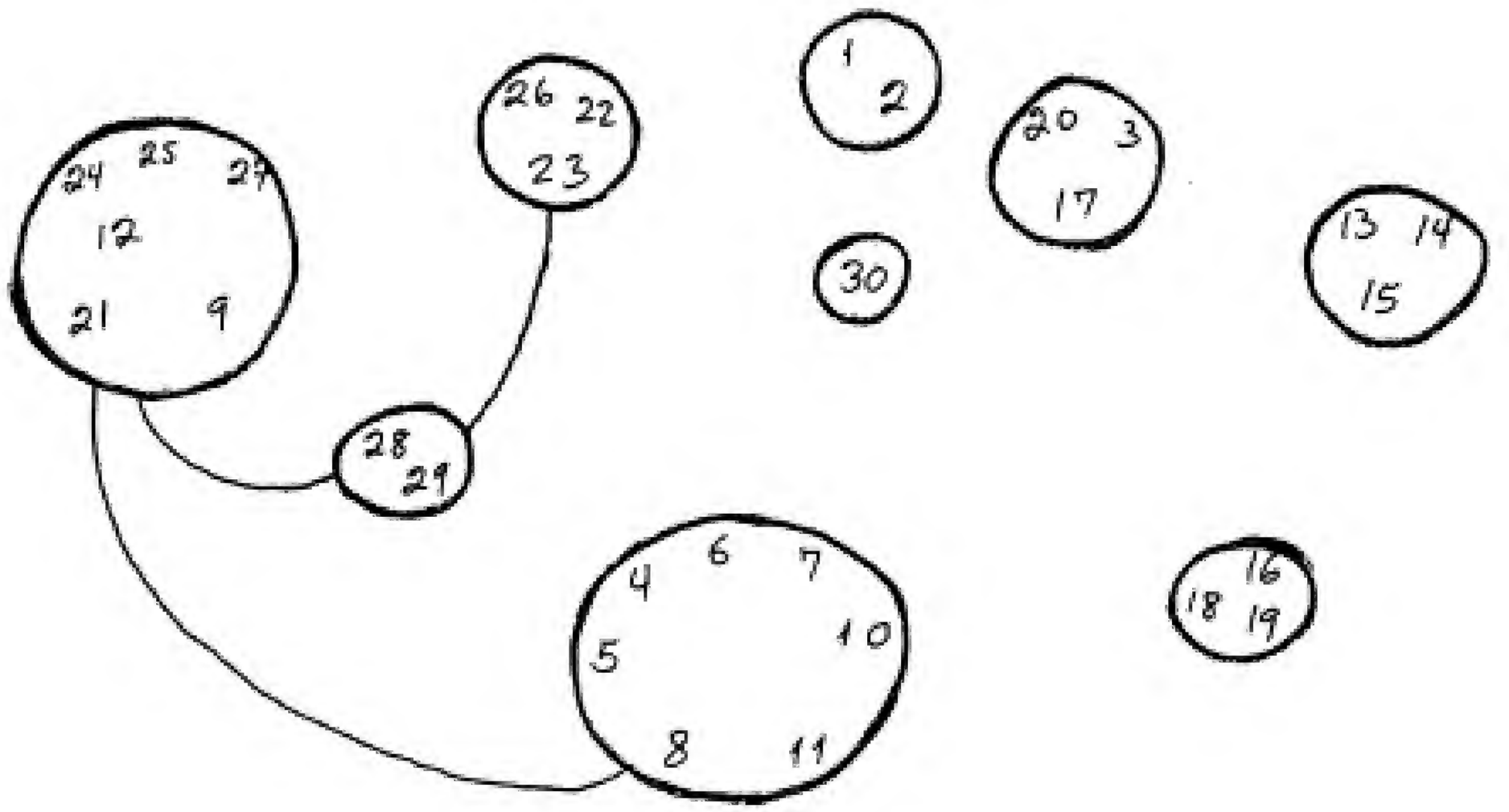


Figure 'F I N A L'.
SINGLEBODIES joints the lonely
region :16 with the nucleus :18-19.

The definition of the function (SINGLEBODIES R) is given now, where R is a list of nuclei , each one having the form $((R_1\ R_2\ R_3\ \ldots)\ L_1\ L_2\ L_3\ L_4\ \ldots)$; the Rs are regions forming a given nucleus, and the Ls are links (gensyms) emanating from that nucleus.

```
(DEFPROP SINGLEBODIES (LAMBDA (A)
  (AND (NULL (CDDR A))  (NULL (CDAR A))  (SETQ ** (CADR A))
       (SETQ **
             (FHAVING
              (FUNCTION
               (LAMBDA (X) (AND (NOT (EQ X A)) (MEMBER ** (CDR X)))))
              R))
       (SETQ ** (CAR **))
       (PRINT (QJ (SINGLEBODY  ASSUMES  (EV (CAR **))  (EV (CAR A))
                  SAME  BODY)))
       (NCONC (CAR **) (CAR A))  (RPLACA A ())  (RPLACD A ())))
EXPR)
```

(FHAVING P L), where P is a predicate and L a list, returns a one-element list containing the first element of L which satisfies P.

RESULTS.- Screening out the regions that belong to the background, the nuclei are printed as "bodies."

```
RESULTS  (PRINT (QUOTE RESULTS))  (SETQ J 0)
(MAPC (FUNCTION
        (LAMBDA (J)
                (OR (NULL (CAR J))  (MEMBER (CAAR J) BACKGROUND)
                    ((LAMBDA (BASE)
                             (PRINT (QU (BODY (EV (SETQ J
                                                        (ADD1 J)))))
                                    IS  (EV* (CAR J)))) W.Y
                     10.))))
        R)
```

In this process, we ignore the links which may be joining some of the nuclei : RESULTS considers the links of figure 'FINAL', for instance, non-existent.

DISCUSSION

Division of the evidence in two kinds, strong and weak, seems to be a good compromise. Since SEE requires two strong evidences to join two nuclei , I believe it will lie in the "safe" side, that is, SEE will almost never join two regions that belong to different bodies. Rather, the mistakes of SEE are of the other kind: some times, regions that should be joined are left separated.

When there is ambiguity in the scene, SEE behaves conservatively. We will have an opportunity to see this in the results of the analysis of some of the scenes that follow.

The purpose of the 'weak' links is to reinforce, if necessary, the strong links.

More heuristics could be added to increase the number of links; in particular, given a good number of "link proposers", parameters set outside (by the user) would tell SEE which set of heuristics to use; for instance, if we knew that the scene is formed by prisms, we could use the heuristics that ask for parallel lines having a given configuration, we could check the length of certain edges, etc. None of this is incorporated in the actual SEE program.

SOME INTERESTING EXAMPLES

We present in this section several scenes with the results as computed by SEE.

Example. 'CORN'.- The program analyzes the scene CORN (see fig. CORN) obtaining the following identification of bodies:

```
(SINGLEBODY ASSUMES (:2 :3 :1) (:4) SAME BODY)
(BODY 1. IS :2 :3 :1 :4)
(BODY 2. IS :7 :6 :5)
(BODY 3. IS :13 :11 :12)
(BODY 4. IS :15 :16 :14)
(BODY 5. IS :19 :18 :17)
(BODY 6. IS :21 :20)
(BODY 7. IS :8 :9 :10)
NIL
```

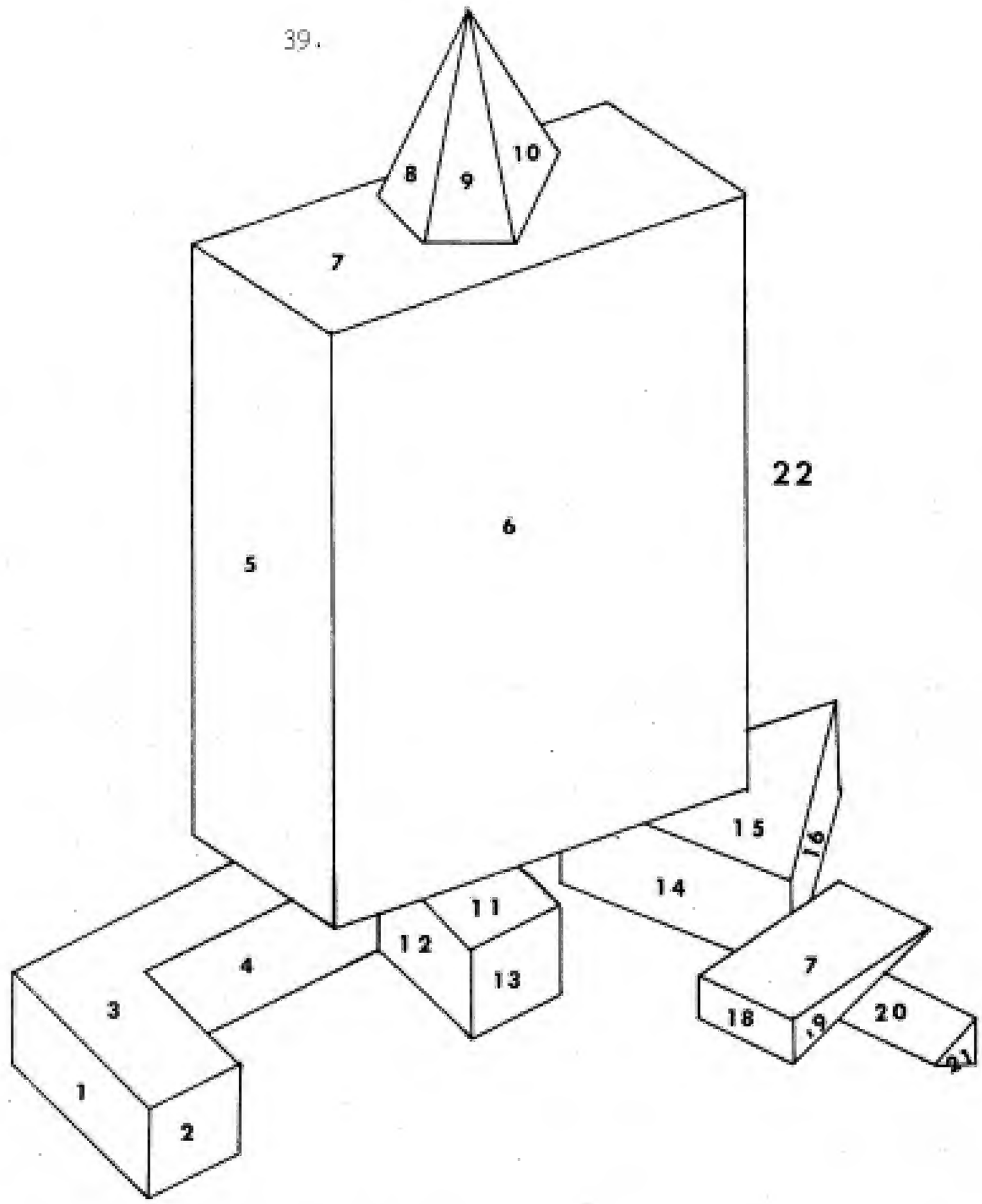
Results for 'CORN'

The region :4 got a single link with :3, and SINGLEBODY had to join it with the nucleus :1 :2 :3. Note that :4 and :12 did not get joined. The pyramid at the top was easily identified because its peak produces a lot of links.

Example. 'MOMO'.- (see scene 'MOMO') The results are as follows:

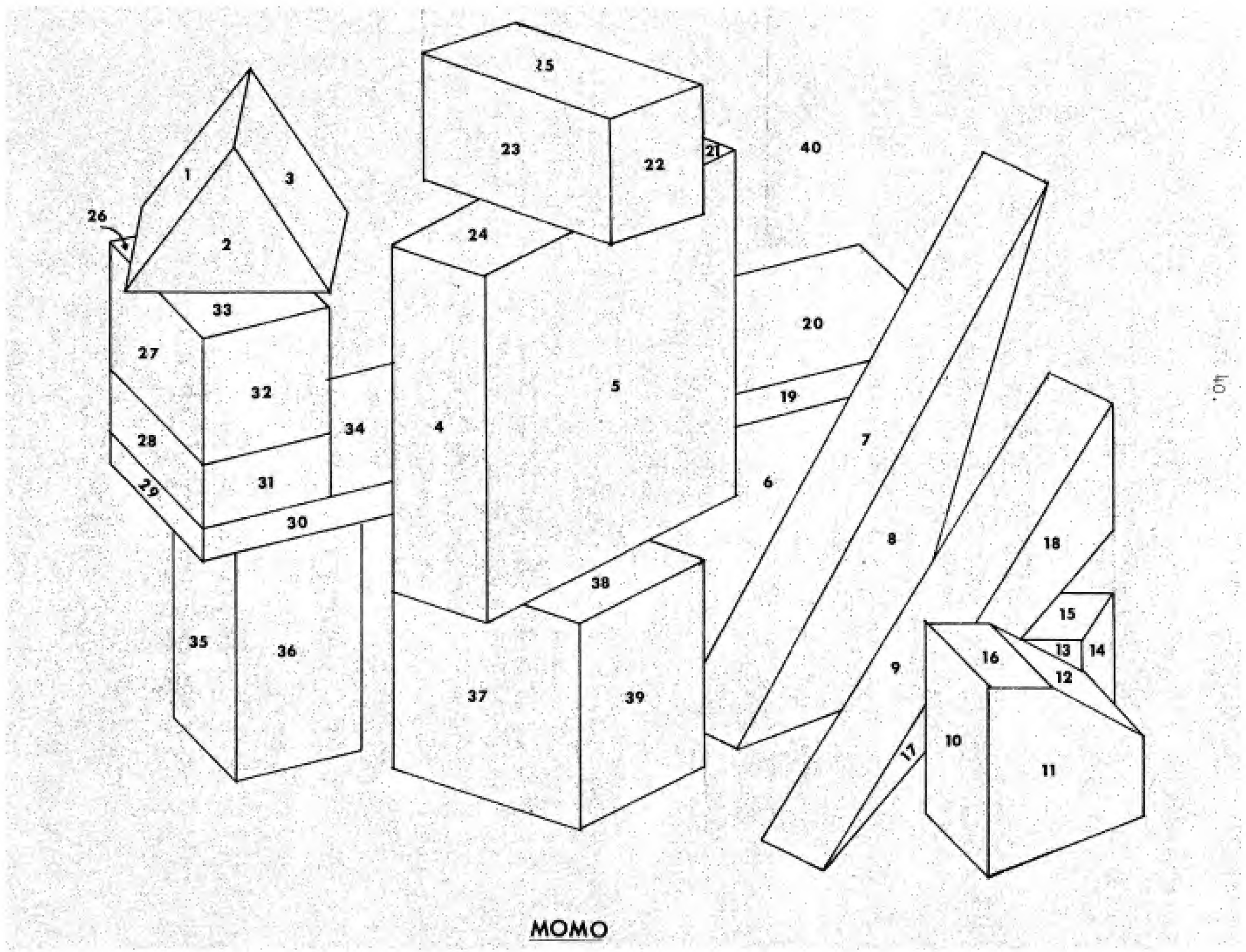
```
(LOCAL ASSUMES (:17) (:9) SAME BODY)
(LOCAL ASSUMES (:9 :17) (:18) SAME BODY)
(BODY 1. IS :3 :2 :1)
(BODY 2. IS :32 :33 :27 :26)
(BODY 3. IS :28 :31)
(BODY 4. IS :19 :20 :34 :30 :29)
(BODY 5. IS :36 :35)
(BODY 6. IS :24 :5 :21 :4)
(BODY 7. IS :25 :23 :22)
(BODY 8. IS :14 :13 :15)
(BODY 9. IS :10 :16 :11 :12)
(BODY 10. IS :18 :9 :17)
(BODY 11. IS :7 :8)
(BODY 12. IS :38 :37 :39)
NIL
```

Results for 'MOMO'.



CORN

Since a link between :4 and :12 is
not established, the bodies found
in that part of the scene are
:1 :2 :3 :4 and :11 :12 :13.

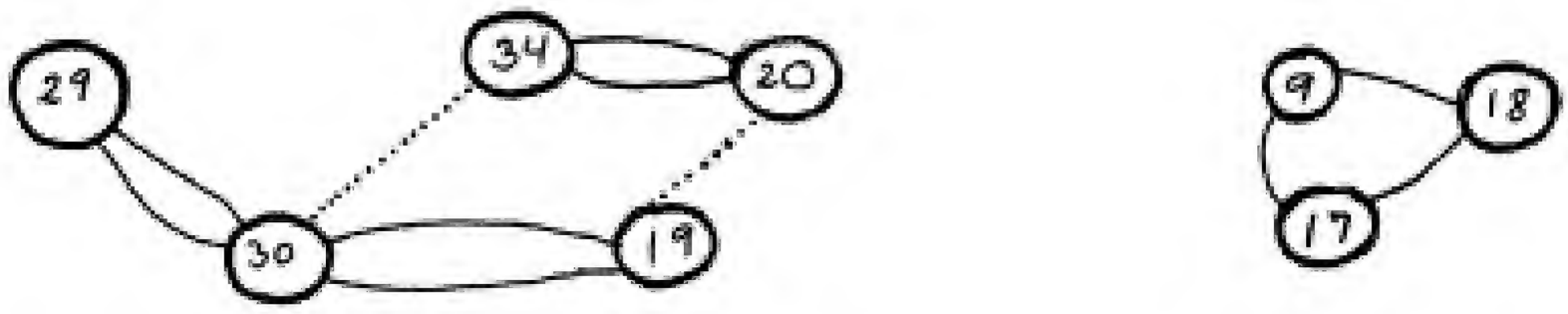

15
23
22
21
40
1
3
26
2
24
33
20
27
32
5
19
34
4
7
28
31
6
29
30
8
18
38
15
35
36
16
13
14
9
12
37
39
10
17
11

MOMO

Comments on the solution for 'MOMO': The central cubes are easily isolated. :21 gets a link with :5 and another with :24, so there is no need to use LOCAL in it. The same is true of :26 with :27 and :33.

There is enough strong evidence to joint :28 with :31.

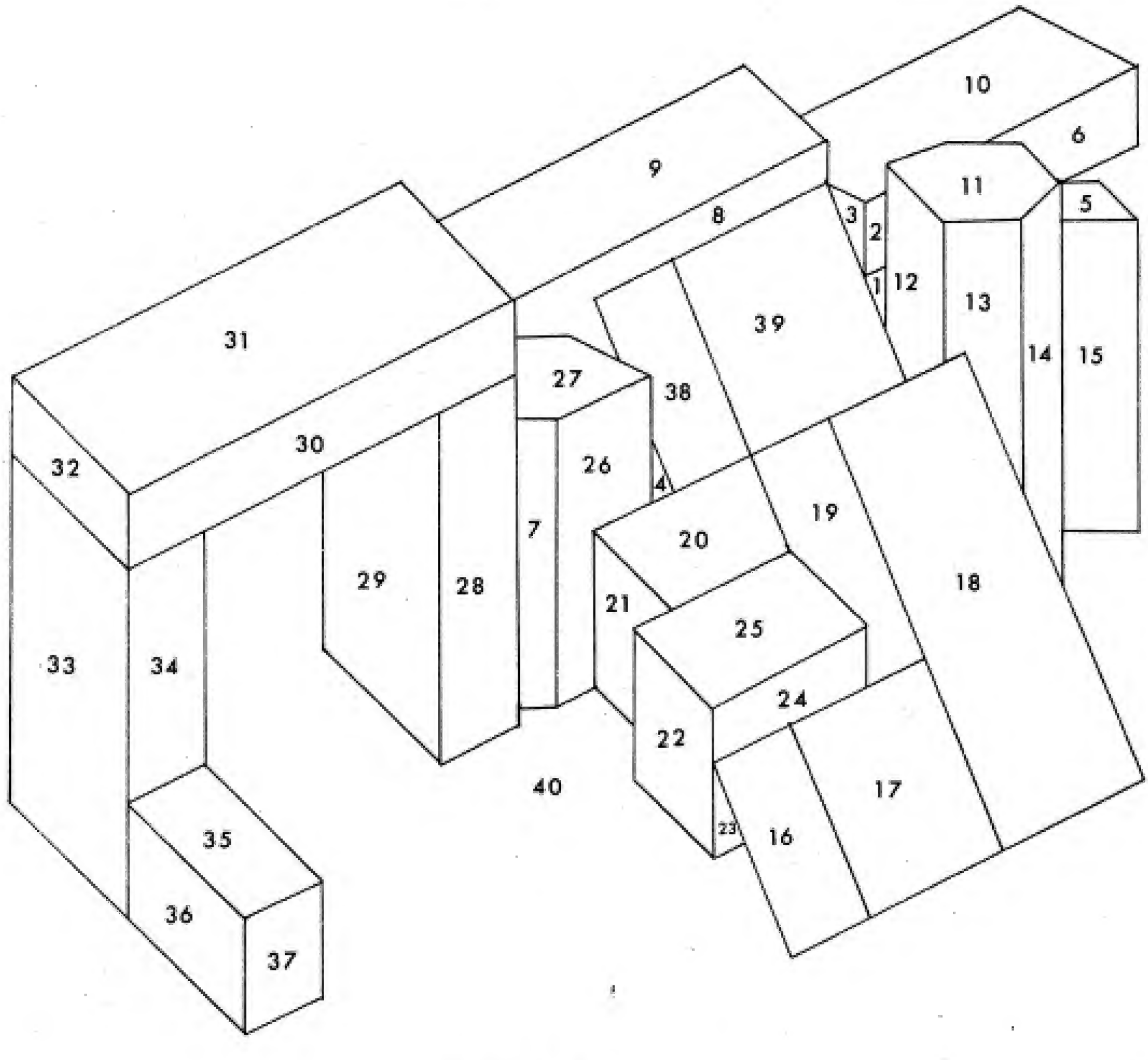
The links for :29 :30 :34 :20 :19 and :9 :17 :18 are as follows:



The dotted links are due to the heuristic of parallel lines between regions :30 and :34, and between :20 and :19 (see page 40). These links make possibble to joint the otherwise disconnected nuclei :29- :30- :19 and :34- :20. In particular, if :34 or :20 did not have parallel sides, SEE would have failed to merge these nuclei , and would have reported two bodies. The disposition of regions in MOMO is such that no link is present between :29 and :34, since :28 and :31 fall "exactly" parallel to :29 and :30. In a less extreme scene, some of these links would be present, allowing correct identification even if :29-:30-:34-:20-:19 were not a prism. Anyway, SEE did not do any mistakes.

The triangle of links formed by :9, :18 and :17 normally would have produced 3 separate bodies (since we need at least one double link to merge two regions); nevertheless, in this case, LOCAL makes some assumptions and saves the situation. Again, if :9 were not a parallelogram, there would be no weak links between :9 and :18 or between :9 and :17, and 3 bodies would have been reported instead of :9- :17-:18. But we should notice that, in general, two links instead of one would be between :17 and :18.

Links were established between :12 and :13, without bad consequences. That is because we need two mistakes, two wrong strong links, to fool to the program. But that could happen.



HOME

The hexagonal prisms did not cause trouble;
SINGLEBODY was needed to group :34 with :33.
The body :38-39 was identified as a single one, but :16-17 was reported as two.
Note that there does not exist local link between :21 and :20; nevertheless, SINGLEBODY makes the correct identification.

Example. 'H O M E'.- (see scene HOME). The results are:

```
(SINGLEBODY ASSUMES (:38) (:39) SAME BODY)
(SINGLEBODY ASSUMES (:34) (:33) SAME BODY)
(SINGLEBODY ASSUMES (:25 :24 :22) (:23) SAME BODY)
(SINGLEBODY ASSUMES (:20) (:21) SAME BODY)
RESULTS
(BODY 1. IS :3 :6 :10 :2)
(BODY 2. IS :14 :13 :11 :12)
(BODY 3. IS :9 :8)
(BODY 4. IS :18)
(BODY 5. IS :15 :5)
(BODY 6. IS :38 :39)
(BODY 7. IS :7 :26 :27)
(BODY 8. IS :20 :21)
(BODY 9. IS :29 :28)
(BODY 10. IS :34 :33)
(BODY 11. IS :31 :32 :30)
(BODY 12. IS :25 :24 :22 :23)
(BODY 13. IS :16)
(BODY 14. IS :19)
(BODY 15. IS :17)
(BODY 16. IS :36 :37 :35)
NIL
```

Results for HOME

Comments on the solution to HOME: There is certain degree of ambiguity in this scene, which human beings tend to resolve by assuming parallelepipeds. :16 and :17 are reported as two separate bodies, instead of one, which I think is a more natural answer.

In the picture from which 'HOME' was drawn, :19 and :18 were the two faces of a single parallelepiped leaning on :38-:39; SEE reports :19 as one body and :18 as another.

Nevertheless, SEE reports :38 and :39 as forming part of the same body (which was in fact the case in the picture in question), due to the fact that :4 and :1 are background.

Example. 'S P R E A D'.- The results are:

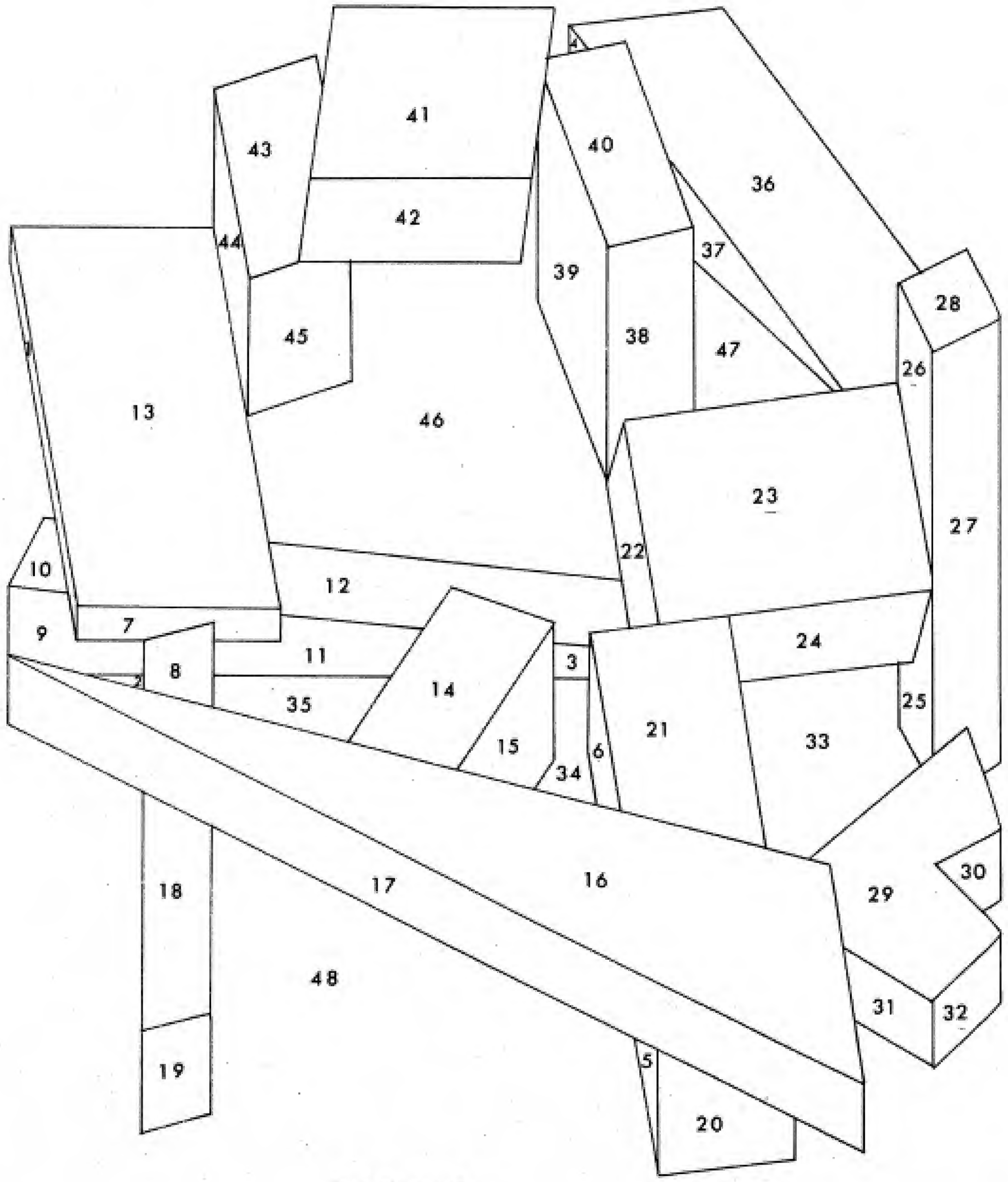
```
(LOCAL ASSUMES (:36) (:4) SAME BODY)
(LOCAL ASSUMES (:30) (:31 :32 :29) SAME BODY)
(LOCAL ASSUMES (:16) (:17) SAME BODY)
(LOCAL ASSUMES (:5) (:20) SAME BODY)
(LOCAL ASSUMES (:3 :11 :9) (:12 :10) SAME BODY)

(SINGLEBODY ASSUMES (:23) (:22) SAME BODY)
(SINGLEBODY ASSUMES (:4 :36) (:37) SAME BODY)
(SINGLEBODY ASSUMES (;28 :26 :27) (:25) SAME BODY)
(SINGLEBODY ASSUMES (:8) (:18) SAME BODY)
RESULTS
(BODY 1. IS :39 :40 :38)
(BODY 2. IS :23 :22)
(BODY 3. IS :42 :41)
(BODY 4. IS :4 :36 :37)
(BODY 5. IS :24)
(BODY 6. IS :28 :26 :27 :25)                    Results for SPREAD
(BODY 7. IS :31 :32 :29 :30)
(BODY 8. IS :20 :5)
(BODY 9. IS :12 :10 :3 :11 :9)
(BODY 10. IS :13 :7 :1)
(BODY 11. IS :21 :6)
(BODY 12. IS :8 :18)
(BODY 13. IS :17 :16)
(BODY 14. IS :45 :43 :44)
(BODY 15. IS :19)
(BODY 16. IS :15 :14)
NIL
```

Comments on the solution to SPREAD: The body :22-23-24, due to insufficiency of links, was split in two: :22-23 and :24.

Since there is only one link between :6 and :5, this body gets split into two: :6-5 and :21-20. Note that :21 is not the same face as :20, and there is where SEE gets confused and refuses to see evidence toward linking :21 with :20.

The long body :9-10-11-12-3 gets properly identified.



SPREAD

:41 and :42 are identified as a single body. Nevertheless, :8-18-19 gets broken into :8-18 and :19. :28-27-26-25 gets correctly identified, as same as the funny looking :29-30-31-32.

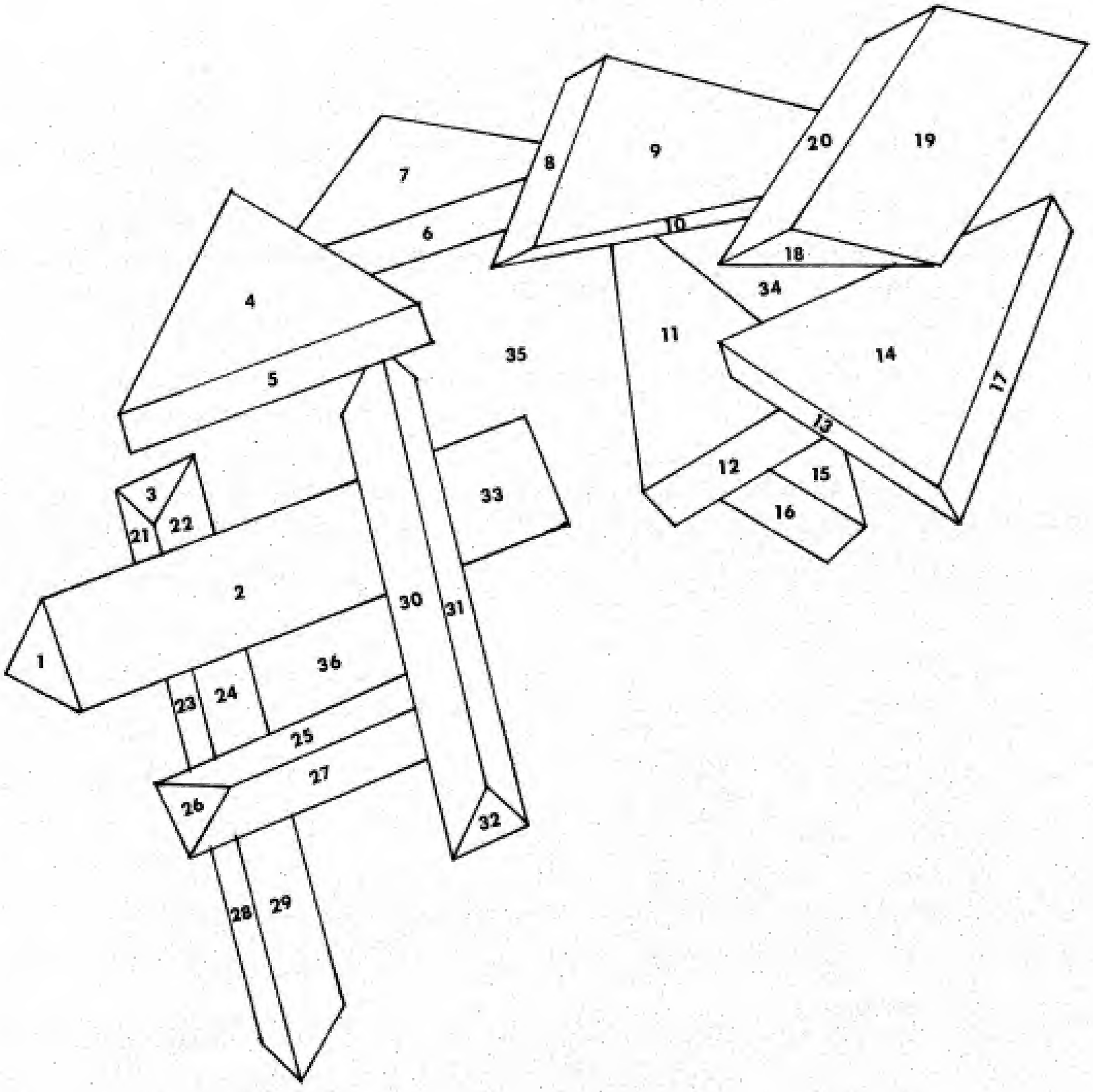
Example. 'H A R D'.- This scene is analyzed with the following results:

```
(LOCAL ASSUMES (:11) (:12) SAME BODY)
(LOCAL ASSUMES (:15) (:16) SAME BODY)
RESULTS
(BODY 1. IS :12 :11)
(BODY 2. IS :16 :15)
(BODY 3. IS :32 :31 :30)
(BODY 4. IS :9 :10 :8)
(BODY 5. IS :18 :19 :20)
(BODY 6. IS :13 : 17 :14)                    Results for HARD
(BODY 7. IS :5 :4)
(BODY 8. IS :1 :2 :33)
(BODY 9. IS :24 :23 :22 :3 :21 :28 :29)
(BODY 10. IS :25 :26 :27)
(BODY 11. IS :7)
(BODY :12 IS :6)
NIL
```

Comments on the solution to HARD: :15 and :16 have to make use of weak evidence to get joined and recognized as forming part of the same body. Nevertheless, this is not necessary with :28 and :29, becauuse, through a chain of Ts, there is enough evidence in :3, :21 and :22 to join successfully all that long and twice occluded body.

There is one serious bug in this identification: regions :7 and :6 get identified as two separate bodies, and not as a single one, as one would normally do. This is caused by the fact that neigher :7 nor :6 have visible 'useful' vertices, and there are not in them enough parallel lines to use the heuristic of page 31 .

:33 was recognized as part of :1-2, as it should be.



HARD

Body :21-22-3-23-24-28-29 is reported as a single object, which is correct. Also, region :33 gets joined with :1-2, as it should. Nevertheless, regions :7 and :6 get reported as two bodies.

Acknowledgements.

Michael Speciner made valuable suggeations.
Cornelia Sullivan helped to codify most of the scenes.

REFERENCES

1.- Greenblatt, R. and Holloway, J. Sides 21. MAC-M-320 (AI Memo 101). Project MAC, MIT. (August 1966).

2.- Mann, W. F. Finding the edges of a polygon in the field of view. Tech. Report No. 10, Computer Corporation of America; Cambridge, Mass.

3.- Guzmán, A. Scene Analysis Using the Concept of Model. TEchnical Report No. AFCRL-67-0133, Computer Corporation of America; Cambridge, Mass. (Jan 67)

4.- Guzmán, A. A Primitive Recognizer of Figures in a Scene. MAC-M-342 (AI Vision Memo 119), Project MAC, MIT (January 1967).

5.- Guzmán, A. Some Aspects of Pattern Recognition by Computer. M. S. Thesis, E. E. Dept, MIT (February 1967). Also available as a Project MAC Technical Report: MAC-TR-37.

Addendum.

A shorter but improved version of this memorandum has been published:

6.- Guzmán, A. Decomposition of a Visual Scene into Three-Dimensional Bodies. AFIPS Proceedings of the 1968 Fall Joint Computer Conference Thompson Book Co. Washington, D. C.

The application of SEE to stereo information (two scenes, left and right); improvements to deal with noisy (imperfect) input, more powerful heuristics for linking of regions, etc., will be found in a doctoral thesis:

7.- Guzmán, A. Computer Recognition of Three-Dimensional Objects in a Visual Scene. Ph. D. Thesis, Electrical Engineering Department, Massachusetts Institute of Technology, Cambridge, Mass. (end of 1968 or beginning of 1969). Will probably appear, too, as a Project MAC Technical Report